Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Vision

ENCS5343

Assignment No.1

Comparing Simple Smoothing Filters with Advanced Filters on Noisy Images

Prepared by: **Qusay Taradeh** **1212508.**

Instructor: **Dr. Aziz Qaroush.**

Section: **1.**

Date: **Monday, November 18, 2024.**

Table of Contents

Table of Figures

List of Tables

Introduction

In this assignment, we will compare the performance of several smoothing filters—specifically, the Box filter, Gaussian filter, and Median filter—which are often referred to as basic smoothing filters. Additionally, we will include an advanced filter, the Adaptive filter, in our comparisons. The filters will be applied to images that have been artificially noise-added to simulate noisy images. Each filter's output will then be compared to the original clean image to assess its performance.

The results for each filter will vary depending on the filter's design and parameter settings, such as kernel size. Our comparison will focus primarily on each filter's effectiveness in noise removal, evaluating both the quality of the image after denoising and the amount of noise removed. Additionally, we will consider the computational efficiency of each filter, measured by the time required to produce results.

Quantitative comparisons will be based on metrics such as Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR). We used the NumPy and OpenCV libraries in Python for filter implementation and performance evaluation in this assignment.

Experiments And Results

The experiments done with 4 steps as following:

Step 1: Loading Noisy Images

In this step, I searched for natural images and selected two clean images with notable details to clearly observe edges. These images were loaded into the program using the Image module from the PIL library. I then added Salt and Pepper noise to each image using the random_noise function from the SciKit-Image (skimage) library, selecting the "salt and pepper" (s&p) noise mode. The noise levels were adjusted to three settings by amount: 0.01 for low, 0.025 for medium, and 0.05 for high noise. Also, selecting “gaussian” noise mode and the noise levels were adjusted to three settings by mean equals zero and variance (var): 0.01 for low, 0.1 for medium, and 0.5 for high noise. The figures below show each clean image alongside its corresponding noisy versions for each noise level.

Clean Images:

Figure 1: Clean image 1

Figure 2: Clean Image 2

Salt And Pepper Low Noise (amount = 0.01):

Figure 3: Salt and Pepper low noisy image 1

Figure 4: Salt and Pepper low noisy image 2

Salt And Pepper Medium Noise (amount = 0.025):

Figure 5: Salt and Pepper medium noisy image 1

Figure 6: Salt and Pepper medium noisy image 2

Salt And Pepper High Noise (amount = 0.05):

Figure 7: Salt and Pepper high noisy image 1

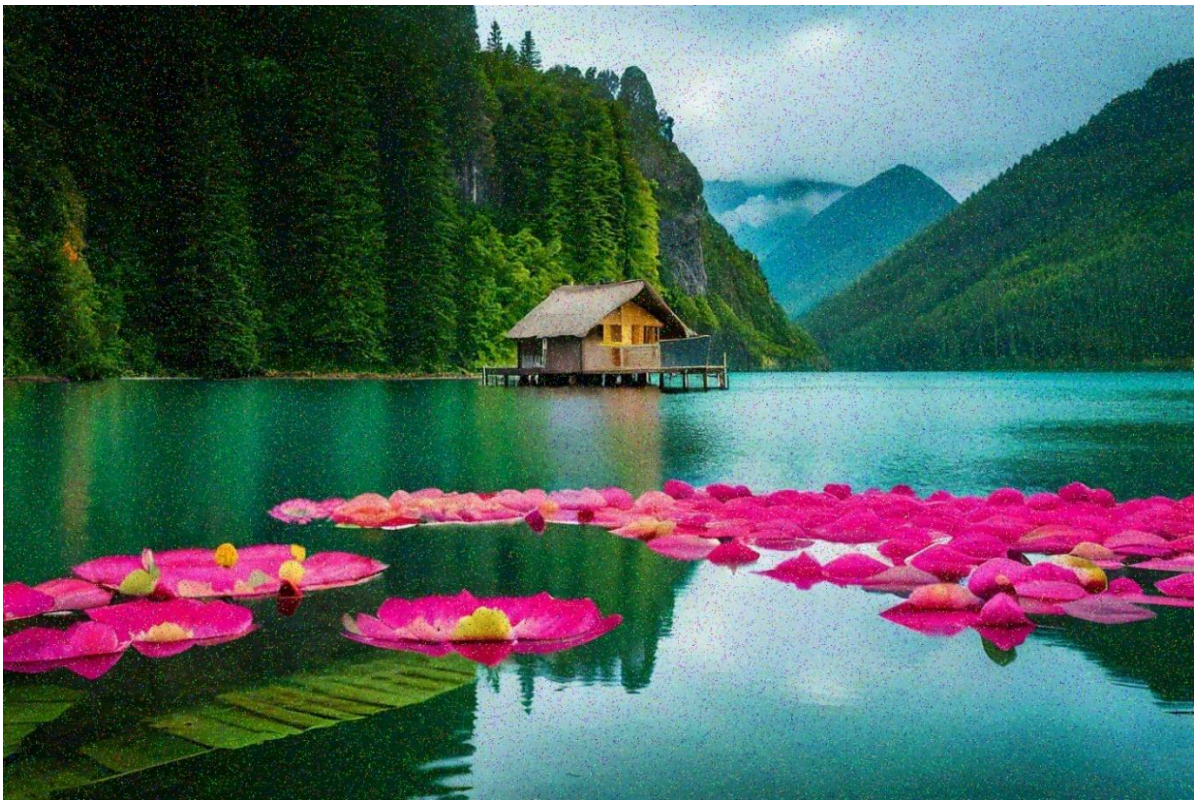


Figure 8: Salt and Pepper high noisy image 2

Gaussian Low Noise (var = 0.01):

Figure 9: Gaussian low noisy image 1

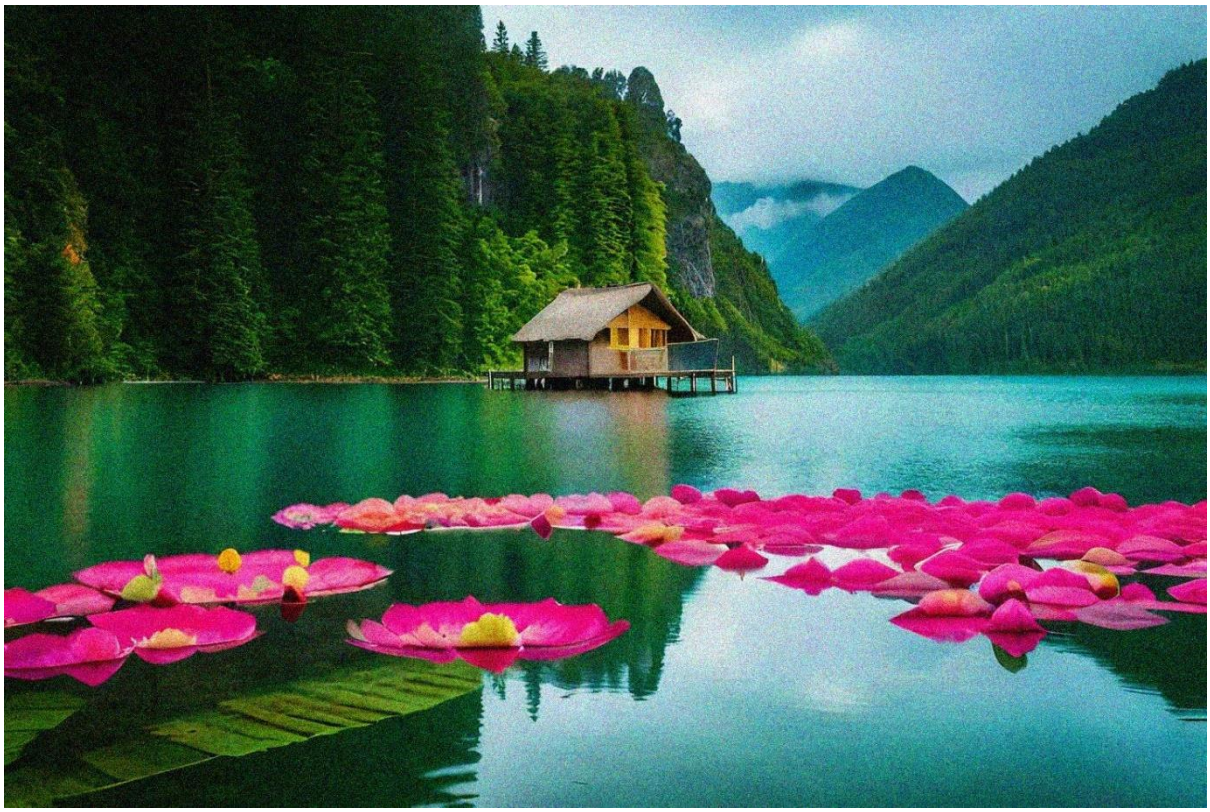


Figure 10: Gaussian low noisy image 2

Gaussian Medium Noise (var = 0.1):

Figure 11: Gaussian medium noisy image 1

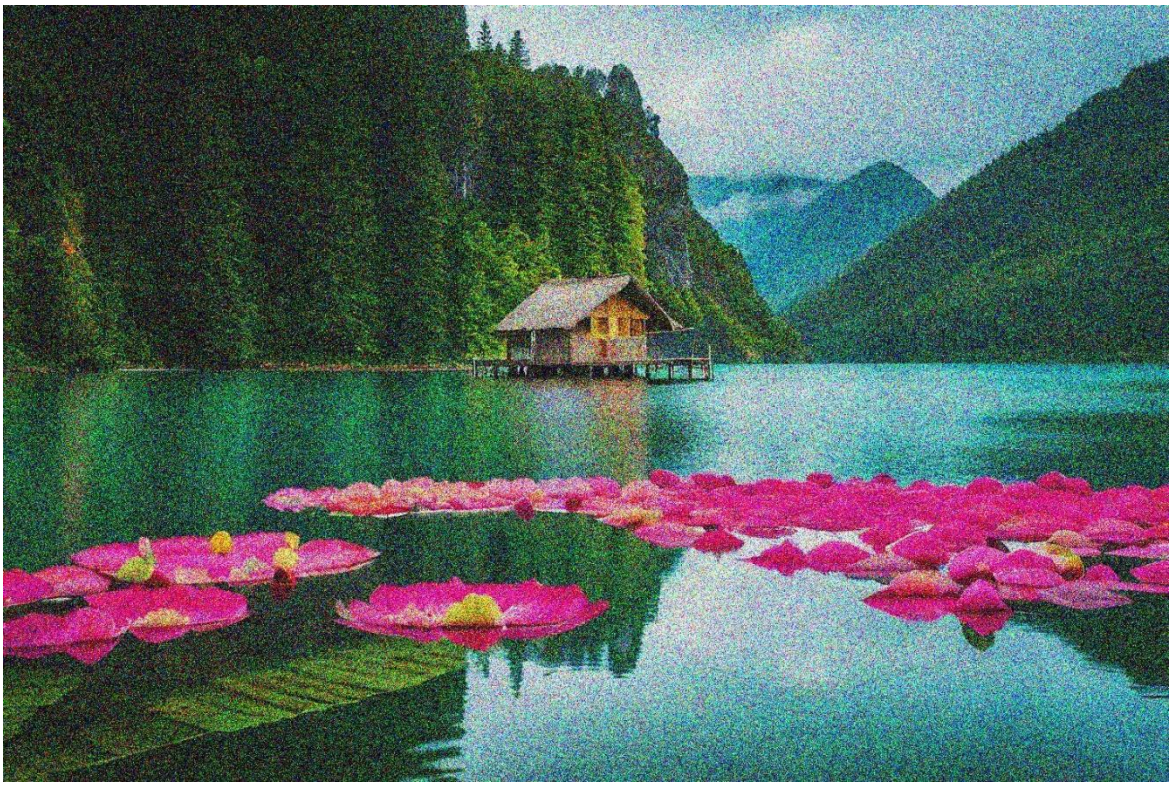


Figure 12: Gaussian medium noisy image 2

Gaussian High Noise (var = 0.5):

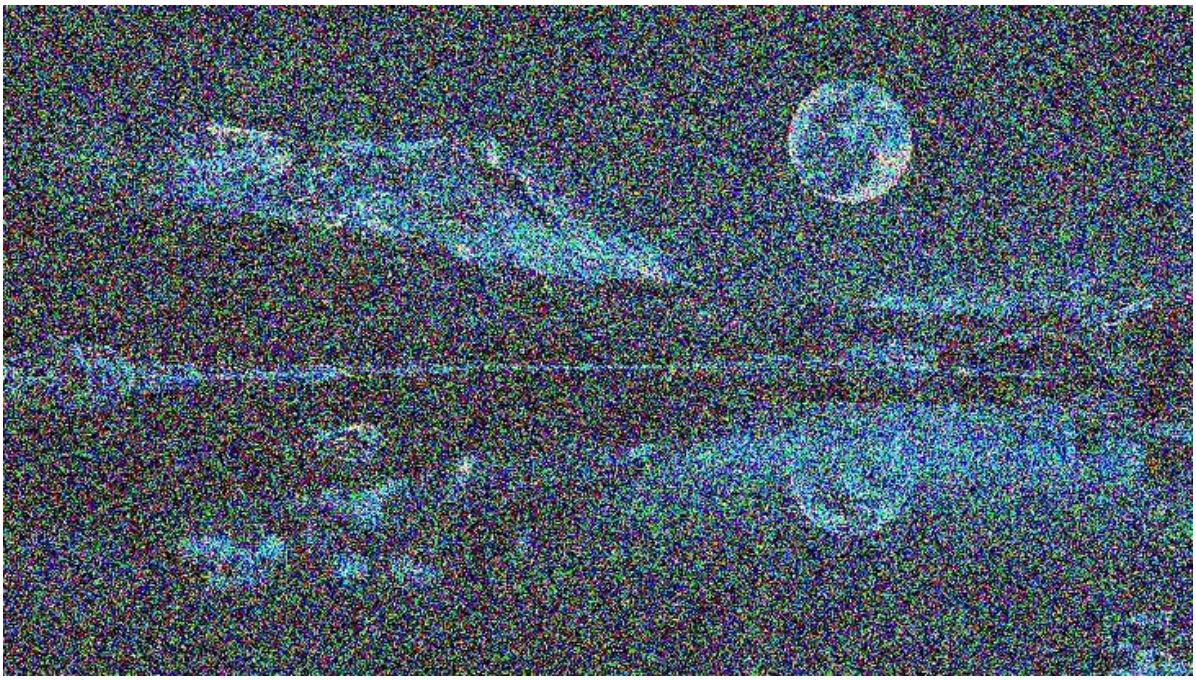


Figure 13: Gaussian high noisy image 1

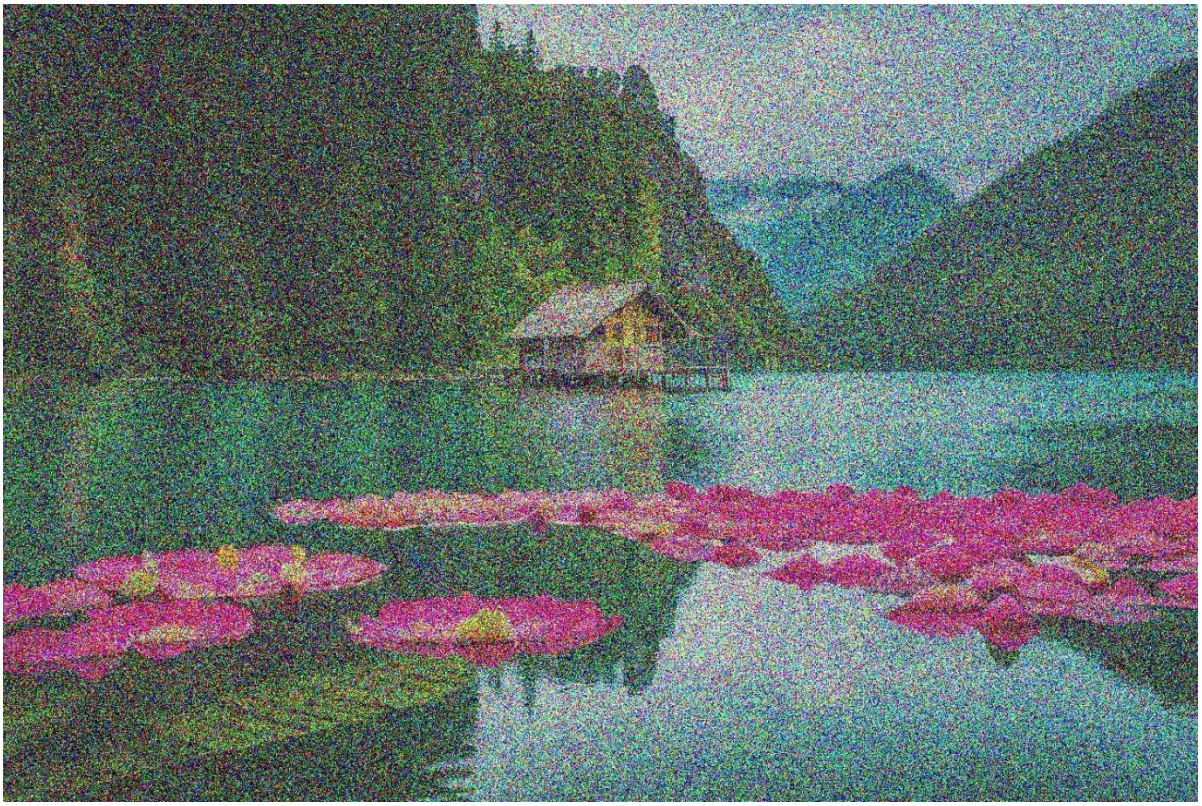


Figure 14: Gaussian high noisy image 2

Step 2: Applying Filters

In this step, I used the OpenCV library to apply built-in functions for each required filter. After converting each image to an array, I passed it through each filter with a customized kernel size for each filter type. This filtering process was applied to images at each noise level.

Box Filter:

The Box filter works by averaging the pixel values within a specified local neighborhood, effectively smoothing out variations in intensity. This averaging process is applied uniformly, meaning each pixel within the kernel contributes equally to the result. While the Box filter is effective at reducing noise, it often causes blurring around edges, as it does not distinguish between noise and important features like edges or fine details. This can result in a loss of sharpness in regions with significant detail or contrast. The filter's smoothing effect is controlled by the kernel size: a larger kernel results in stronger smoothing but more noticeable edge blurring.

Now, we will apply this filter to both noisy images at low, medium, and high noise levels. For each noise level, we will vary the kernel size to observe the changes in the filtered images as follows.

Salt and Pepper Low Noise and Kernel Size = 3:

Figure 15 Output of box filter with k = 3: filtered image 1 that was contained low noise

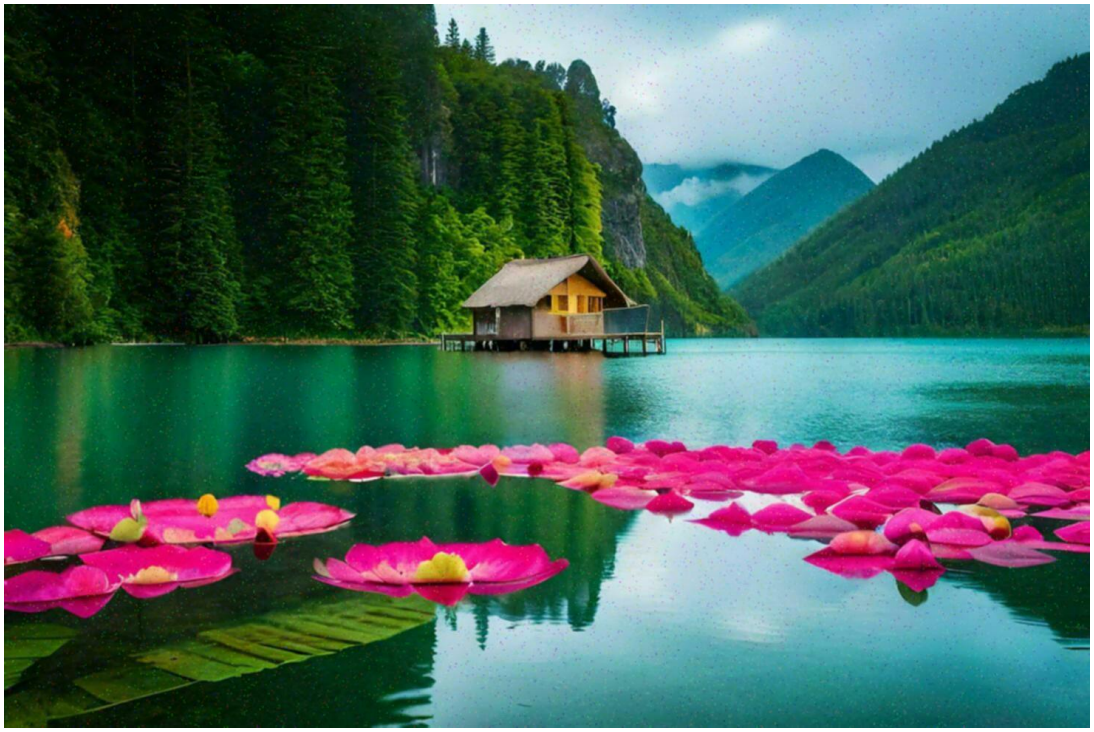


Figure 16 Output of box filter with k = 3: filtered image 2 that was contained low noise

Now, **Kernel Size = 5**

Figure 17 Output of box filter with k = 5: filtered image 1 that was contained low noise

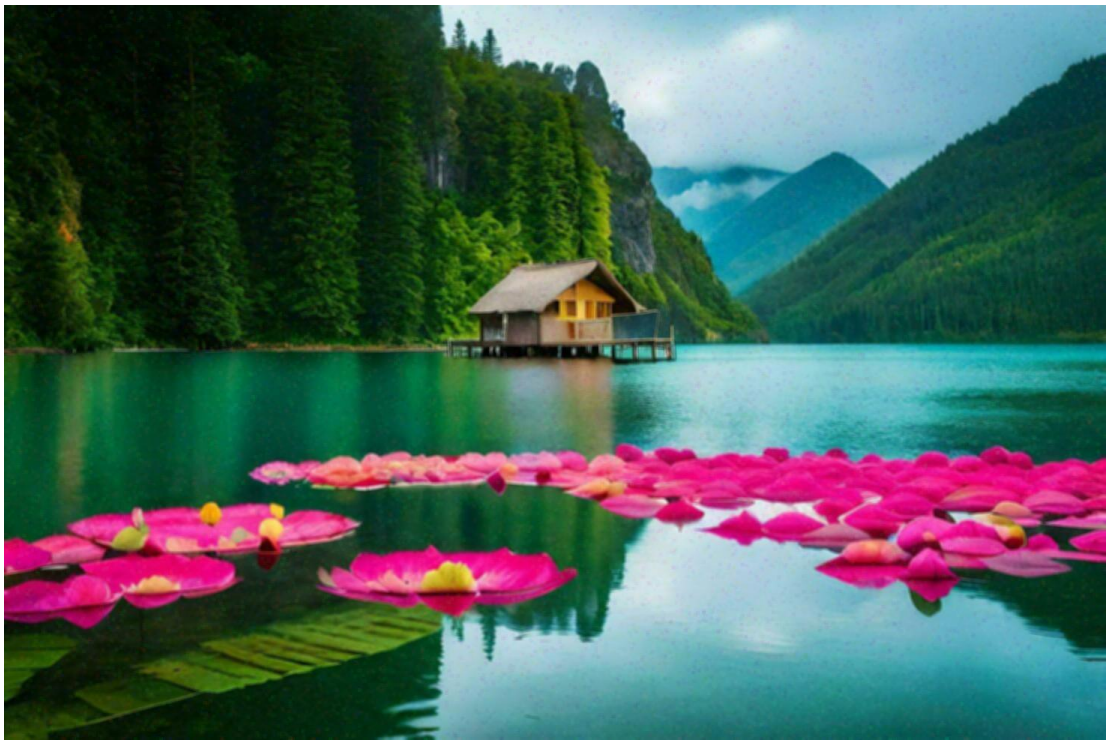


Figure 18 Output of box filter with k = 5: filtered image 1 that was contained low noise s & p

Salt and Pepper Medium Noise and Kernel Size = 3:

Figure 19 Output of box filter with k = 3: filtered image 1 that was contained medium noise s & p

Figure 20 Output of box filter with k = 3: filtered image 2 that was contained medium noise s & p

Now, **Kernel Size = 5**

Figure 21 Output of box filter with k = 5: filtered image 1 that was contained medium noise s & p

Figure 22 Output of box filter with k = 5: filtered image 1 that was contained medium noise s & p

Salt and Pepper High Noise and Kernel Size = 3:

Figure 23 Output of box filter with k = 3: filtered image 1 that was contained high noise s & p

Figure 24 Output of box filter with k = 3: filtered image 2 that was contained high noise s & p

Now, **Kernel Size = 5**

Figure 25 Output of box filter with k = 5: filtered image 1 that was contained high noise s & p

Figure 26 Output of box filter with k = 5: filtered image 1 that was contained high noise s & p

Now, **Kernel Size = 9**

Figure 27 Output of box filter with k = 9: filtered image 1 that was contained high noise s & p

Figure 28 Output of box filter with k = 9: filtered image 1 that was contained high noise s & p

Gaussian Low Noise and Kernel Size = 3:

Figure 29 Output of box filter with k = 3: filtered image 1 that was contained low noise (g)

Figure 30 Output of box filter with k = 3: filtered image 2 that was contained low noise (g)

Now, **Kernel Size = 5**

Figure 31 Output of box filter with k = 5: filtered image 1 that was contained low noise (g)

Figure 32 Output of box filter with k = 5: filtered image 1 that was contained low noise (g)

Gaussian Medium Noise and Kernel Size = 3:

Figure 33 Output of box filter with k = 3: filtered image 1 that was contained medium noise (g)

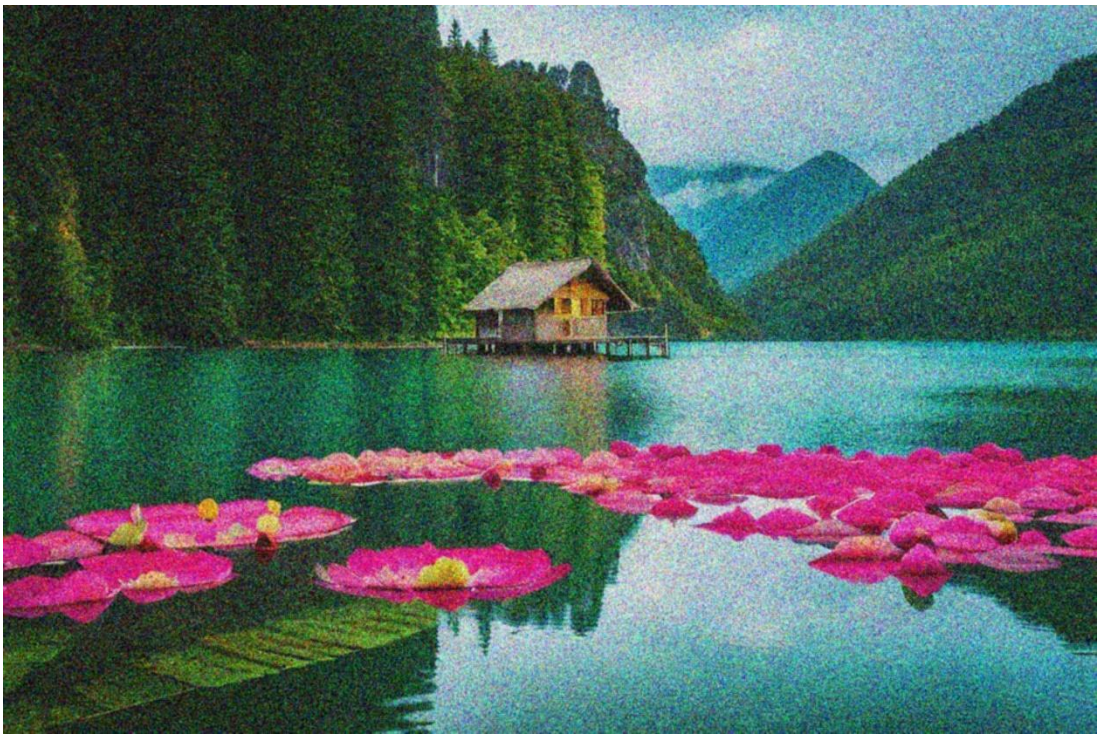


Figure 34 Output of box filter with k = 3: filtered image 2 that was contained medium noise (g)

Now, **Kernel Size = 5**

Figure 35 Output of box filter with k = 5: filtered image 1 that was contained medium noise (g)

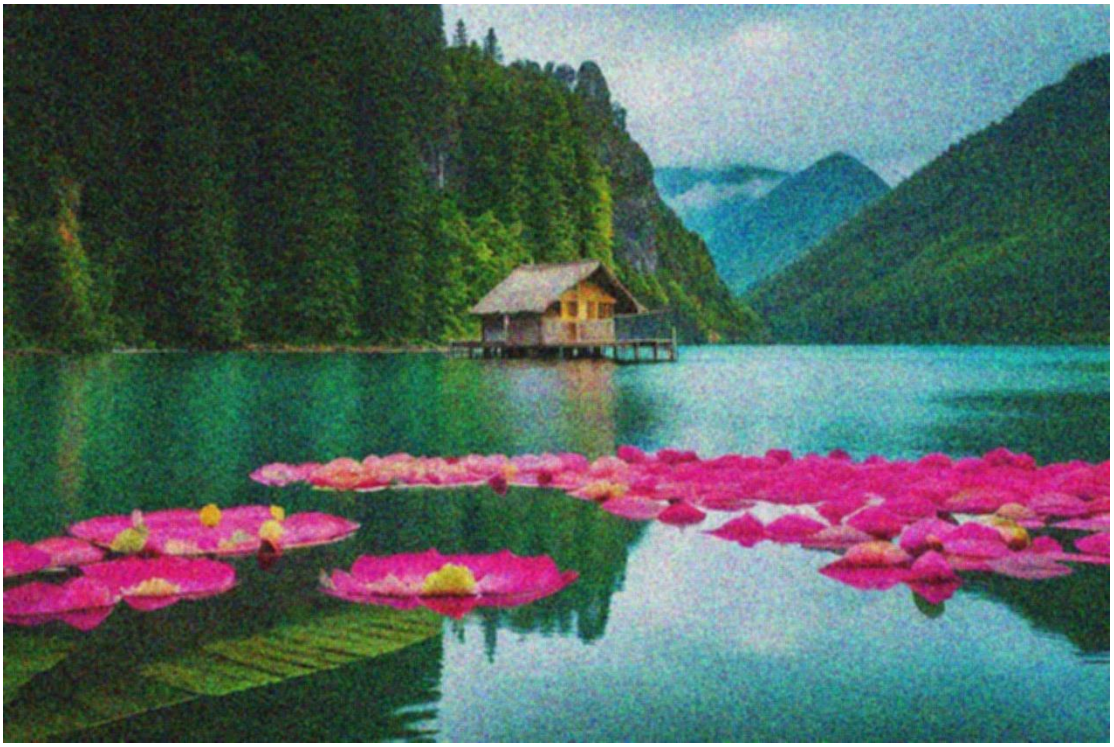


Figure 36 Output of box filter with k = 5: filtered image 1 that was contained medium noise (g)

Gaussian High Noise and Kernel Size = 3:

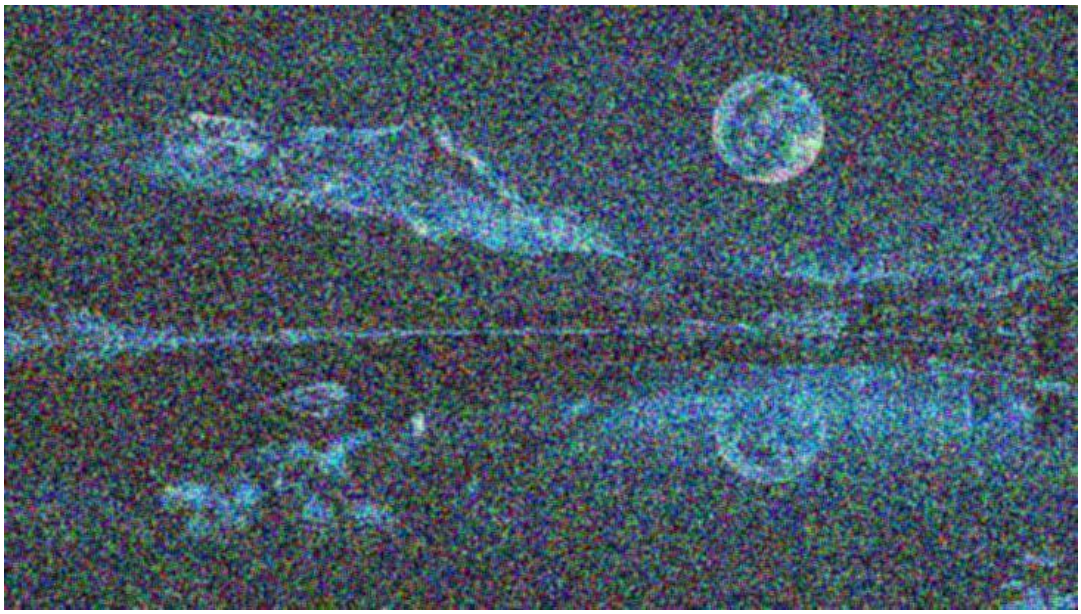


Figure 37 Output of box filter with k = 3: filtered image 1 that was contained high noise (g)

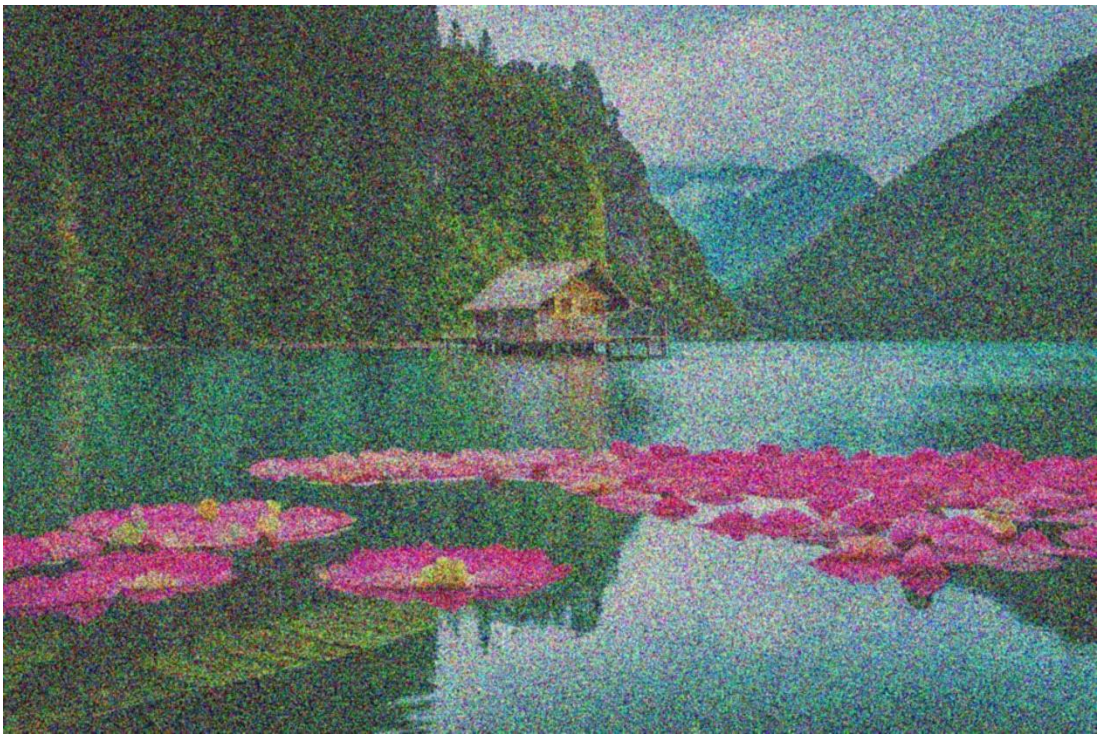


Figure 38 Output of box filter with k = 3: filtered image 2 that was contained high noise (g)

Now, **Kernel Size = 5**

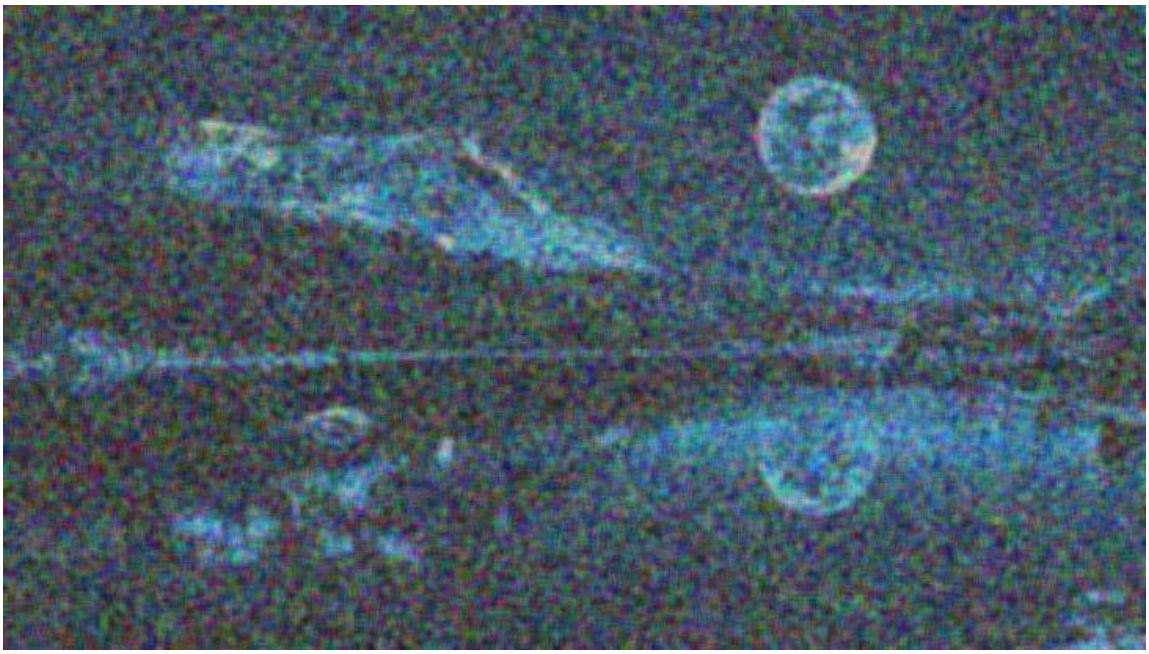


Figure 39 Output of box filter with k = 5: filtered image 1 that was contained high noise (g)

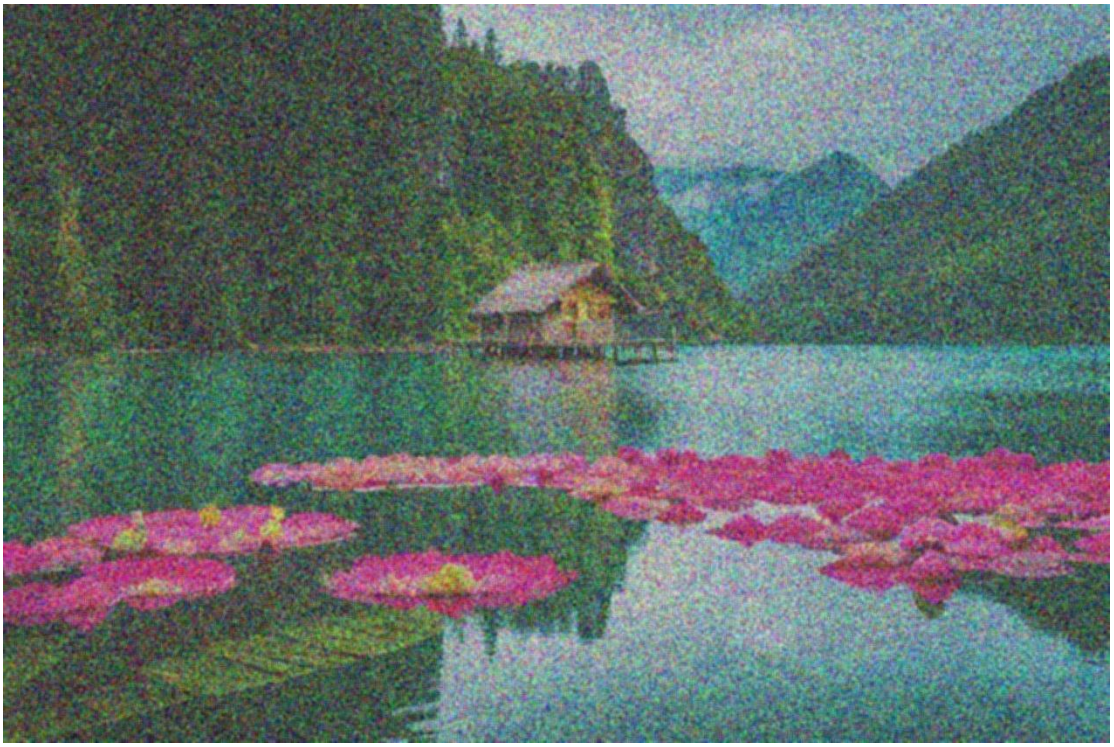


Figure 40 Output of box filter with k = 5: filtered image 1 that was contained high noise (g)

Now, **Kernel Size = 9**

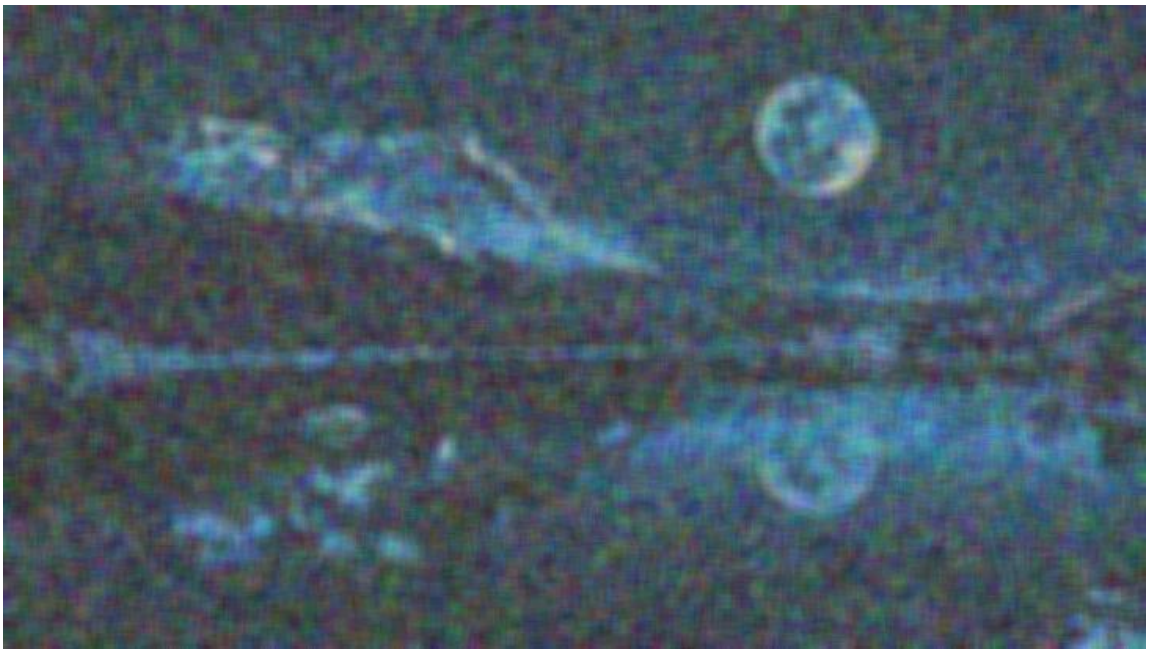


Figure 41 Output of box filter with k = 9: filtered image 1 that was contained high noise (g)

Figure 42 Output of box filter with k = 9: filtered image 1 that was contained high noise (g)

Gaussian Filter:

The Gaussian filter applies a weighted average to each pixel in a local neighborhood, where the weights decrease with distance from the center of the kernel. This weighting scheme is based on a Gaussian distribution, resulting in higher emphasis on pixels closer to the center and progressively lower emphasis on those farther away. By giving more weight to central pixels, the Gaussian filter smooths out noise while preserving edges better than the Box filter, which averages uniformly. The amount of smoothing and the filter's sensitivity to edges can be controlled by adjusting the standard deviation (sigma) and kernel size. A larger sigma or kernel size leads to stronger blurring, which can help reduce noise but may slightly soften edges.

Now, we will apply this filter to both noisy images at low, medium, and high noise levels. For each noise level, we will vary the kernel size to observe the changes in the filtered images as follows.

Salt and Pepper Low Noise and Kernel Size = 3:

Figure 43 Output of gaussian filter with k = 3: filtered image 1 that was contained low noise

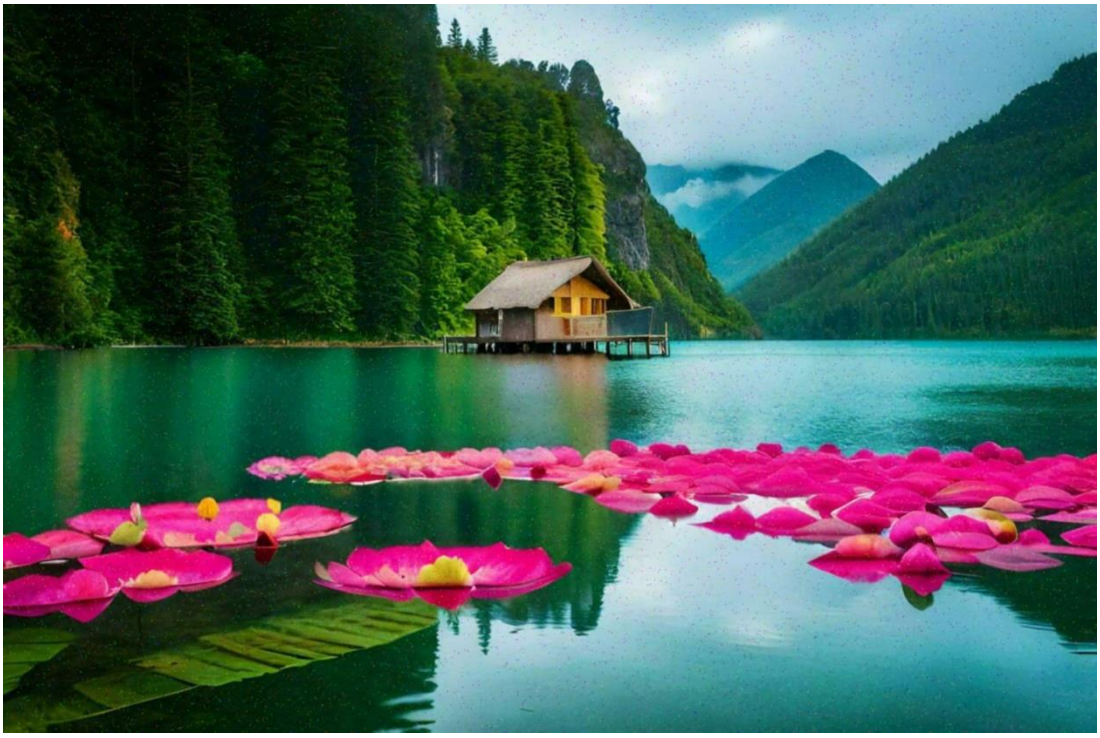


Figure 44 Output of gaussian filter with k = 3: filtered image 2 that was contained low noise

Now, **Kernel Size = 5**

Figure 45 Output of gaussian filter with k = 5: filtered image 1 that was contained low noise

Figure 46 Output of gaussian filter with k = 5: filtered image 1 that was contained low noise s & p

Salt and Pepper Medium Noise and Kernel Size = 3:

Figure 47 Output of gaussian filter with k = 3: filtered image 1 that was contained medium noise s & p

Figure 48 Output of gaussian filter with k = 3: filtered image 2 that was contained medium noise s & p

Now, **Kernel Size = 5**

Figure 49 Output of gaussian filter with k = 5: filtered image 1 that was contained medium noise s & p

Figure 50 Output of gaussian filter with k = 5: filtered image 1 that was contained medium noise s & p

Salt and Pepper High Noise and Kernel Size = 3:

Figure 51 Output of gaussian filter with k = 3: filtered image 1 that was contained high noise s & p

Figure 52 Output of gaussian filter with k = 3: filtered image 2 that was contained high noise s & p

Now, **Kernel Size = 5**

Figure 53 Output of gaussian filter with k = 5: filtered image 1 that was contained high noise s & p

Figure 54 Output of gaussian filter with k = 5: filtered image 1 that was contained high noise s & p

Now, **Kernel Size = 9**

Figure 55 Output of gaussian filter with k = 9: filtered image 1 that was contained high noise s & p

Figure 56 Output of gaussian filter with k = 9: filtered image 1 that was contained high noise s & p

Gaussian Low Noise and Kernel Size = 3:

Figure 57 Output of gaussian filter with k = 3: filtered image 1 that was contained low noise (g)

Figure 58 Output of gaussian filter with k = 3: filtered image 2 that was contained low noise (g)

Now, **Kernel Size = 5**

Figure 59 Output of gaussian filter with k = 5: filtered image 1 that was contained low noise (g)

Figure 60 Output of gaussian filter with k = 5: filtered image 1 that was contained low noise (g)

Gaussian Medium Noise and Kernel Size = 3:

Figure 61 Output of gaussian filter with k = 3: filtered image 1 that was contained medium noise (g)

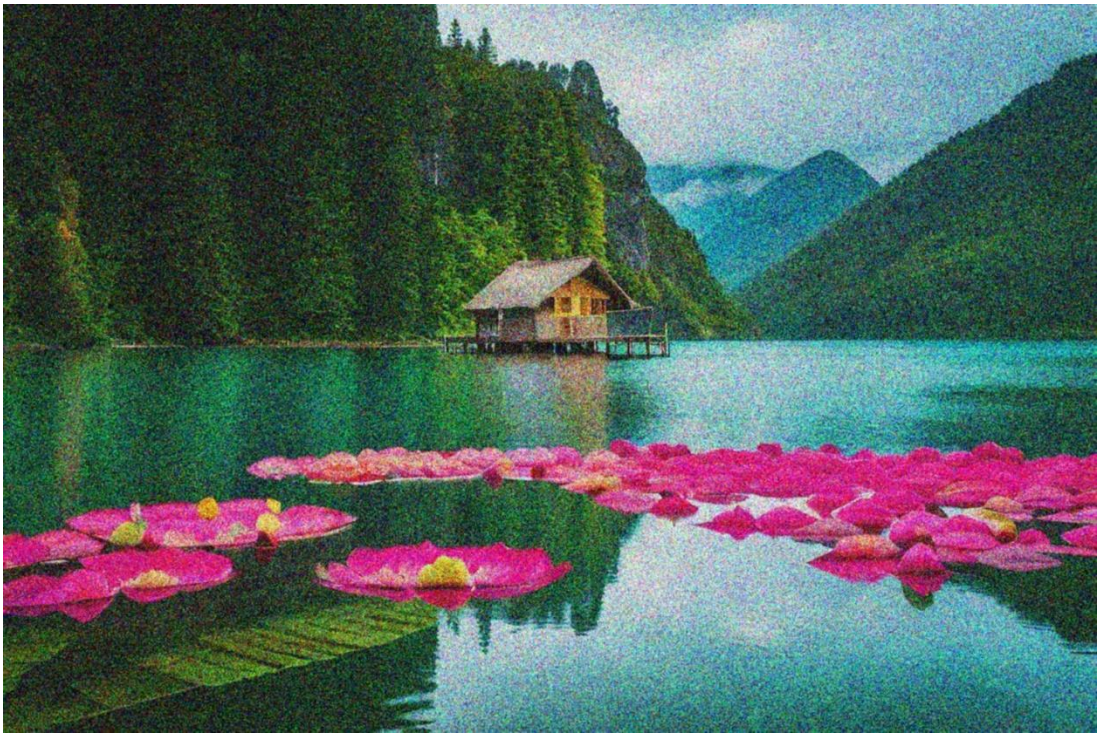


Figure 62 Output of gaussian filter with k = 3: filtered image 2 that was contained medium noise (g)

Now, **Kernel Size = 5**

Figure 63 Output of gaussian filter with k = 5: filtered image 1 that was contained medium noise (g)

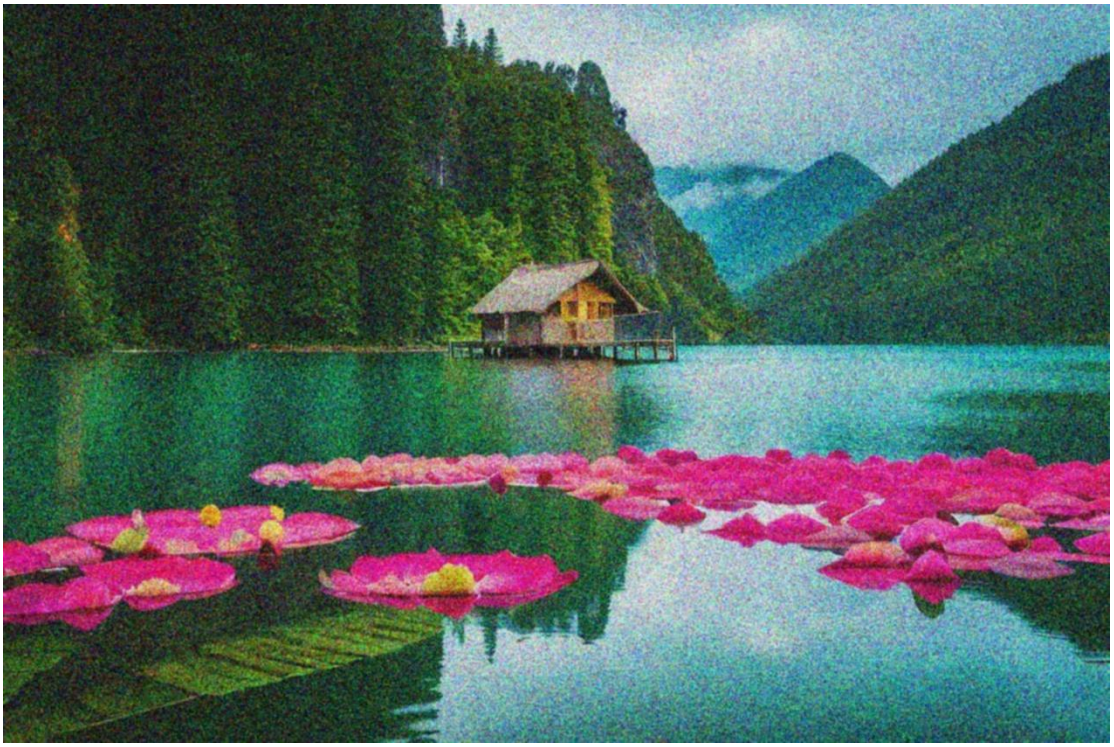


Figure 64 Output of gaussian filter with k = 5: filtered image 1 that was contained medium noise (g)

Gaussian High Noise and Kernel Size = 3:

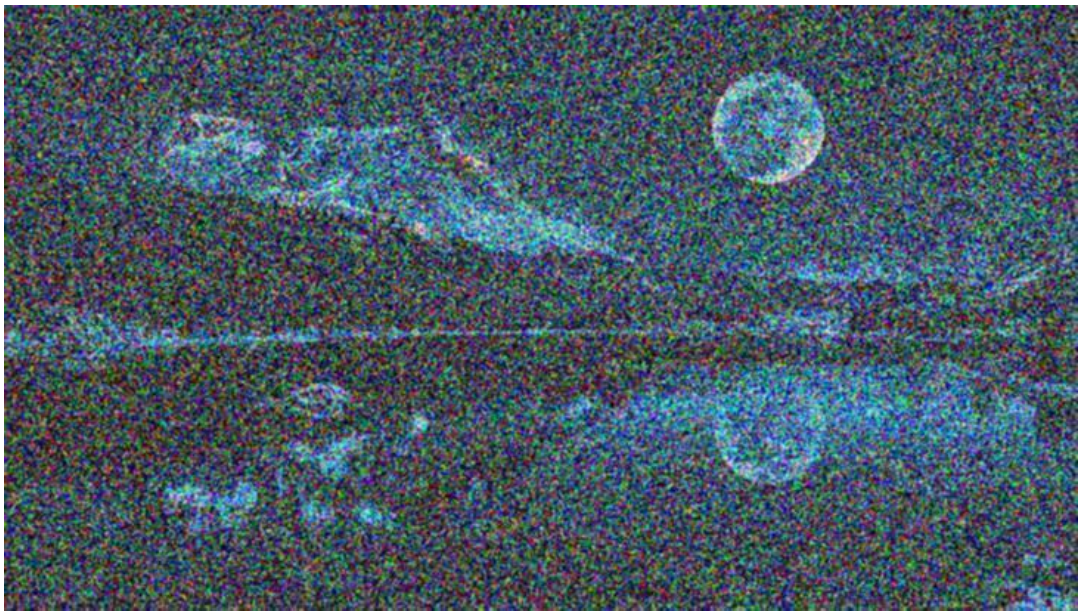


Figure 65 Output of gaussian filter with k = 3: filtered image 1 that was contained high noise (g)

Figure 66 Output of gaussian filter with k = 3: filtered image 2 that was contained high noise (g)

Now, **Kernel Size = 5**

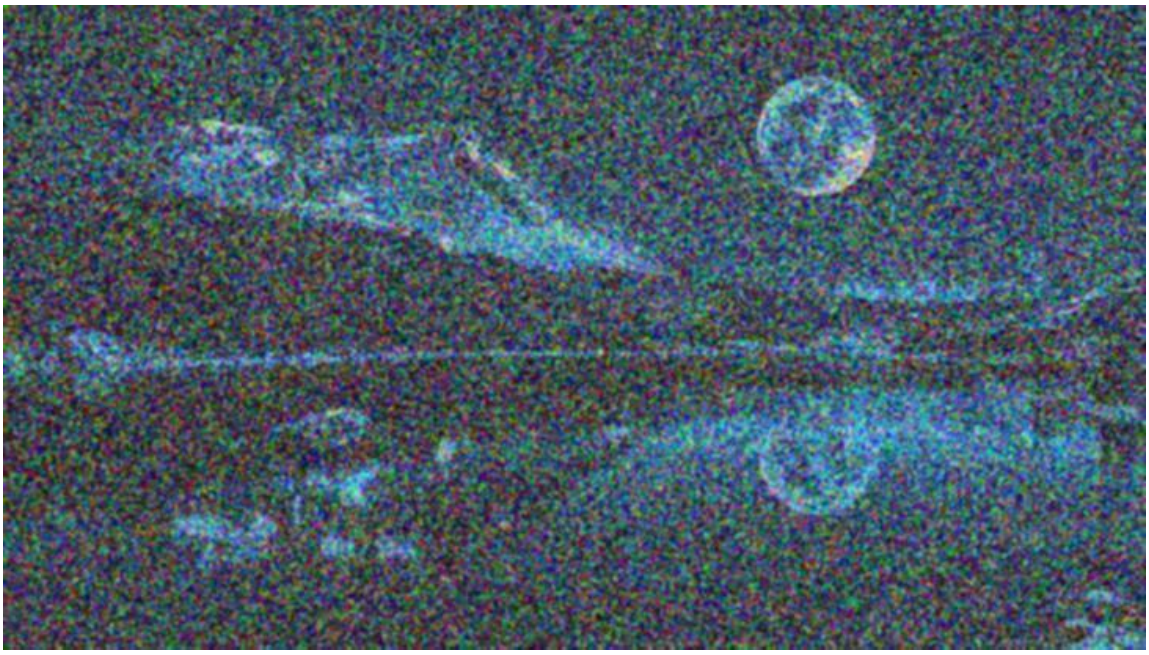


Figure 67 Output of gaussian filter with k = 5: filtered image 1 that was contained high noise (g)

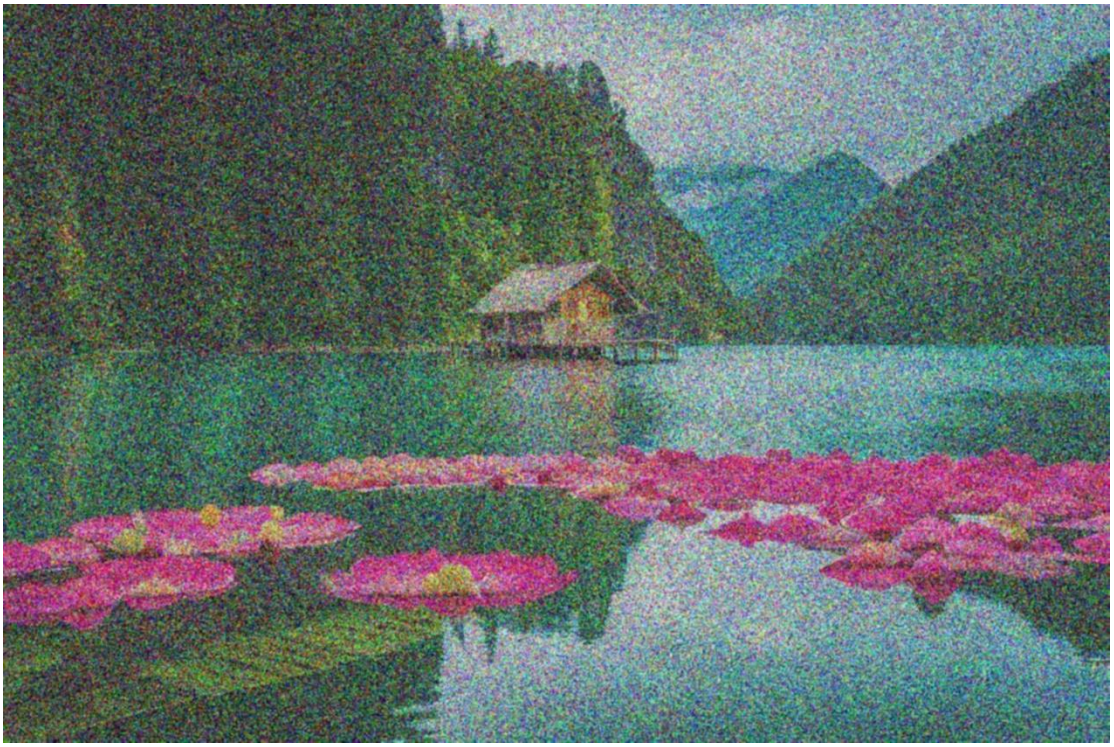


Figure 68 Output of gaussian filter with k = 5: filtered image 1 that was contained high noise (g)

Now, **Kernel Size = 9**

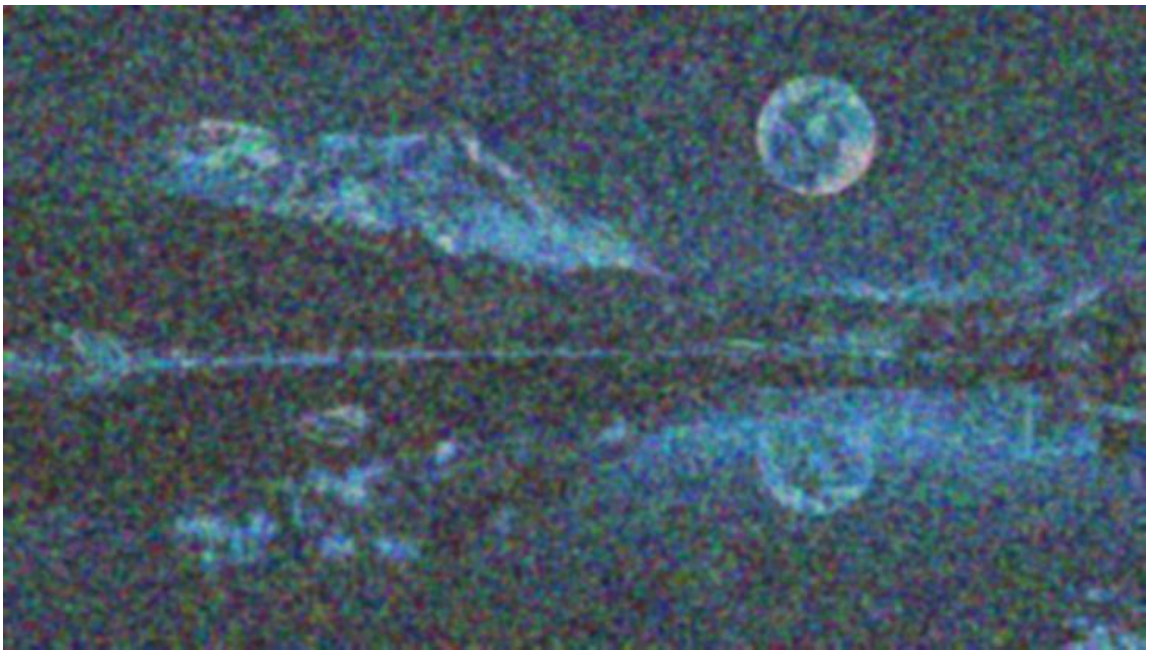


Figure 69 Output of gaussian filter with k = 9: filtered image 1 that was contained high noise (g)

Figure 70 Output of gaussian filter with k = 9: filtered image 1 that was contained high noise (g)

Median Filter:

The Median filter replaces each pixel's value with the median value of its surrounding neighborhood. Unlike averaging filters, which can blur edges, the Median filter is highly effective for removing salt-and-pepper noise because it considers the middle value rather than a weighted or uniform average. This makes it especially useful for images with isolated, high-contrast noise spots, as it can eliminate these noise points without significantly affecting edges or fine details. The Median filter is non-linear, meaning it does not produce a simple weighted sum but instead selects the central value, preserving edge sharpness more effectively than smoothing filters. The effectiveness of the Median filter can vary with the chosen kernel size, where a larger kernel can remove more noise but may slightly reduce detail in densely textured areas.

Now, we will apply this filter to both noisy images at low, medium, and high noise levels. For each noise level, we will vary the kernel size to observe the changes in the filtered images as follows.

Salt and Pepper Low Noise and Kernel Size = 3:

Figure 71 Output of median filter with k = 3: filtered image 1 that was contained low noise s & p

Figure 72 Output of median filter with k = 3: filtered image 2 that was contained low noise s & p

Now, **Kernel Size = 5**

Figure 73 Output of median filter with k = 5: filtered image 1 that was contained low noise

Figure 74 Output of median filter with k = 5: filtered image 1 that was contained low noise s & p

Salt and Pepper Medium Noise and Kernel Size = 3:

Figure 75 Output of median filter with k = 3: filtered image 1 that was contained medium noise s & p

Figure 76 Output of median filter with k = 3: filtered image 2 that was contained medium noise s & p

Now, **Kernel Size = 5**

Figure 77 Output of median filter with k = 5: filtered image 1 that was contained medium noise s & p

Figure 78 Output of median filter with k = 5: filtered image 1 that was contained medium noise s & p

Salt and Pepper High Noise and Kernel Size = 3:

Figure 79 Output of median filter with k = 3: filtered image 1 that was contained high noise s & p

Figure 80 Output of median filter with k = 3: filtered image 2 that was contained high noise s & p

Now, **Kernel Size = 5**

Figure 81 Output of median filter with k = 5: filtered image 1 that was contained high noise s & p

Figure 82 Output of median filter with k = 5: filtered image 1 that was contained high noise s & p

Now, **Kernel Size = 9**

Figure 83 Output of median filter with k = 9: filtered image 1 that was contained high noise s & p

Figure 84 Output of median filter with k = 9: filtered image 1 that was contained high noise s & p

Gaussian Low Noise and Kernel Size = 3:

Figure 85 Output of median filter with k = 3: filtered image 1 that was contained low noise (g)

Figure 86 Output of median filter with k = 3: filtered image 2 that was contained low noise (g)

Now, **Kernel Size = 5**

Figure 87 Output of median filter with k = 5: filtered image 1 that was contained low noise (g)

Figure 88 Output of median filter with k = 5: filtered image 1 that was contained low noise (g)

Gaussian Medium Noise and Kernel Size = 3:

Figure 89 Output of median filter with k = 3: filtered image 1 that was contained medium noise (g)

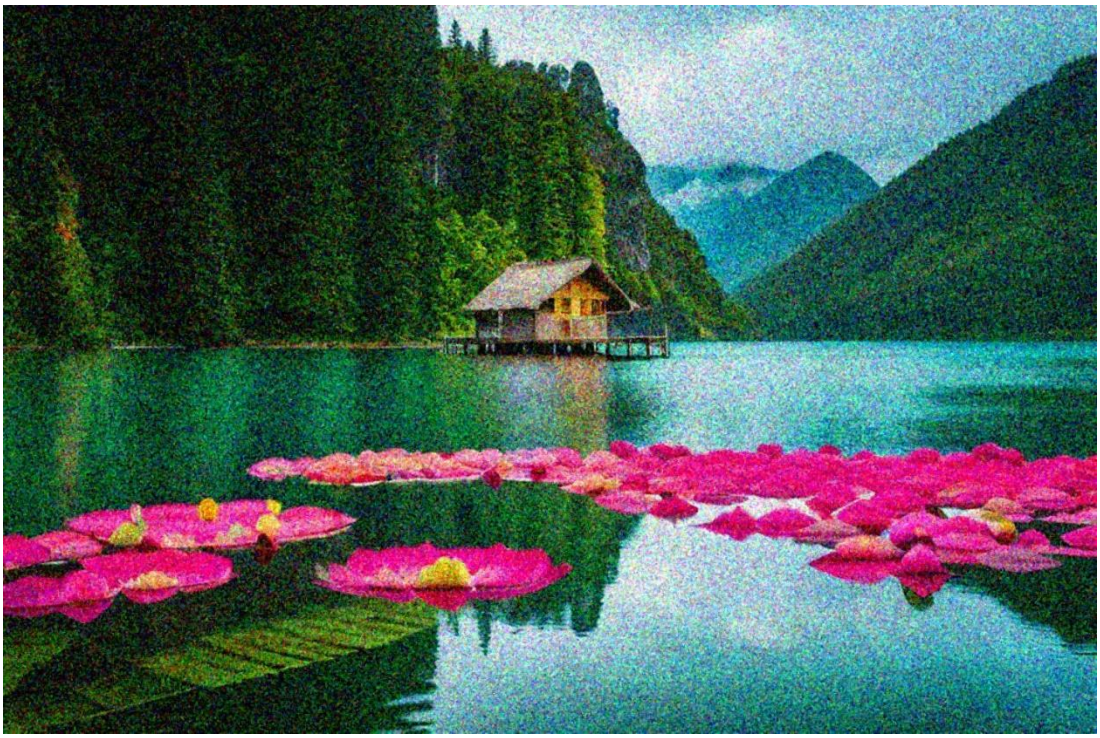


Figure 90 Output of median filter with k = 3: filtered image 2 that was contained medium noise (g)

Now, **Kernel Size = 5**

Figure 91 Output of median filter with k = 5: filtered image 1 that was contained medium noise (g)

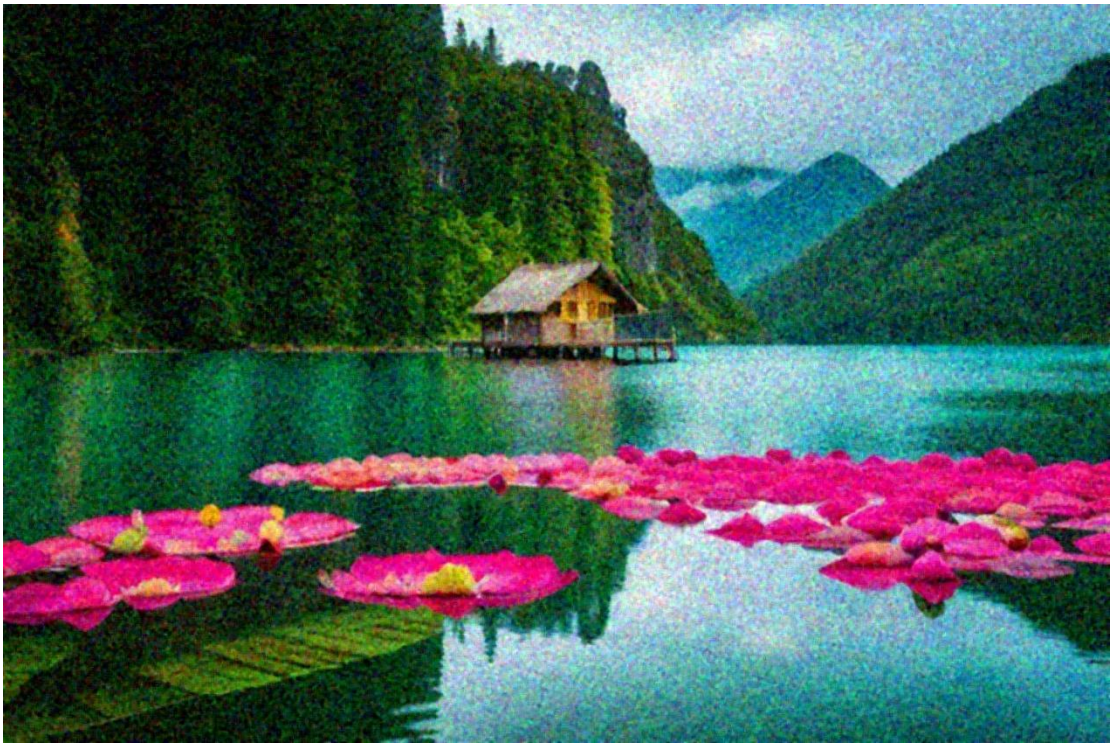


Figure 92 Output of median filter with k = 5: filtered image 1 that was contained medium noise (g)

Gaussian High Noise and Kernel Size = 3:

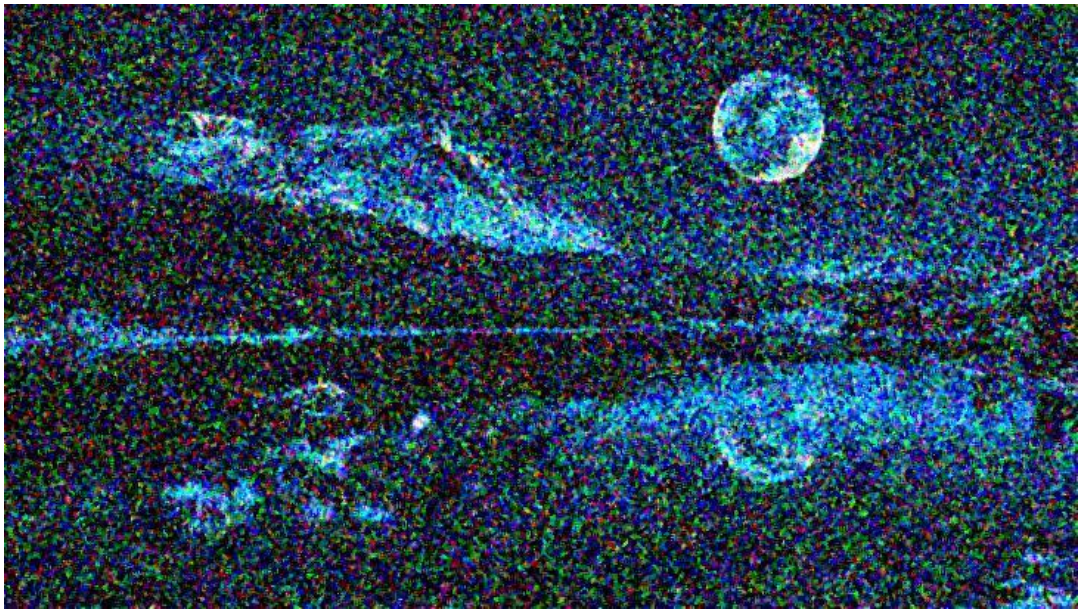


Figure 93 Output of median filter with k = 3: filtered image 1 that was contained high noise (g)

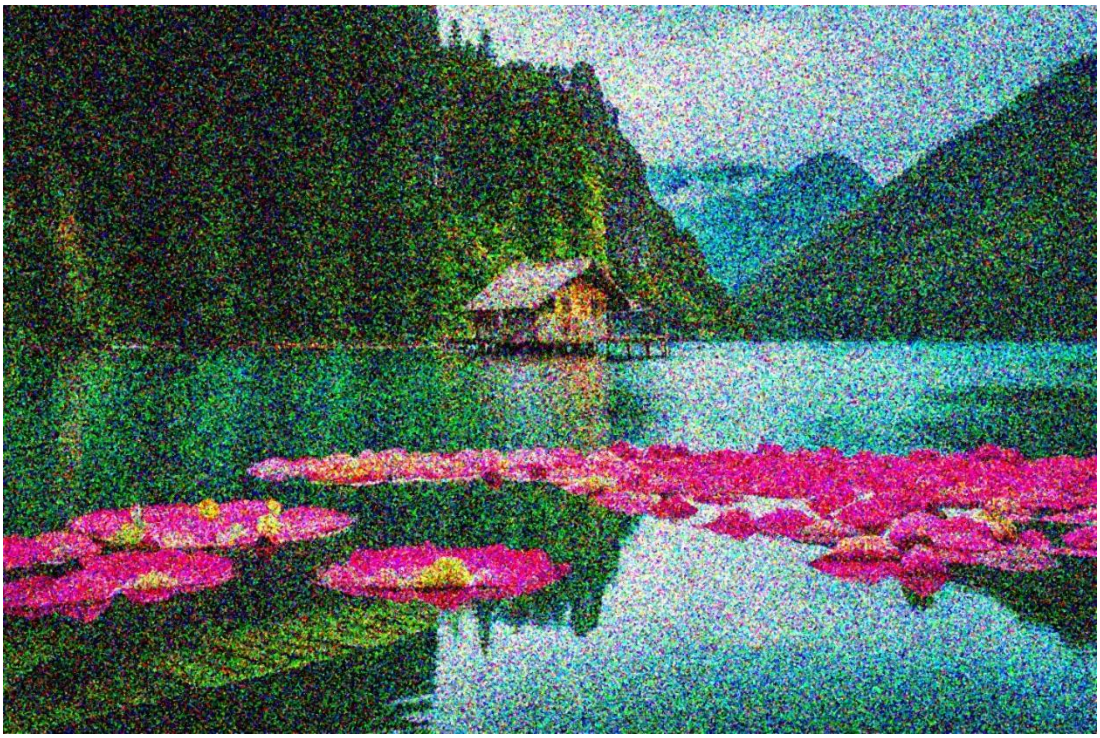


Figure 94 Output of median filter with k = 3: filtered image 2 that was contained high noise (g)

Now, **Kernel Size = 5**

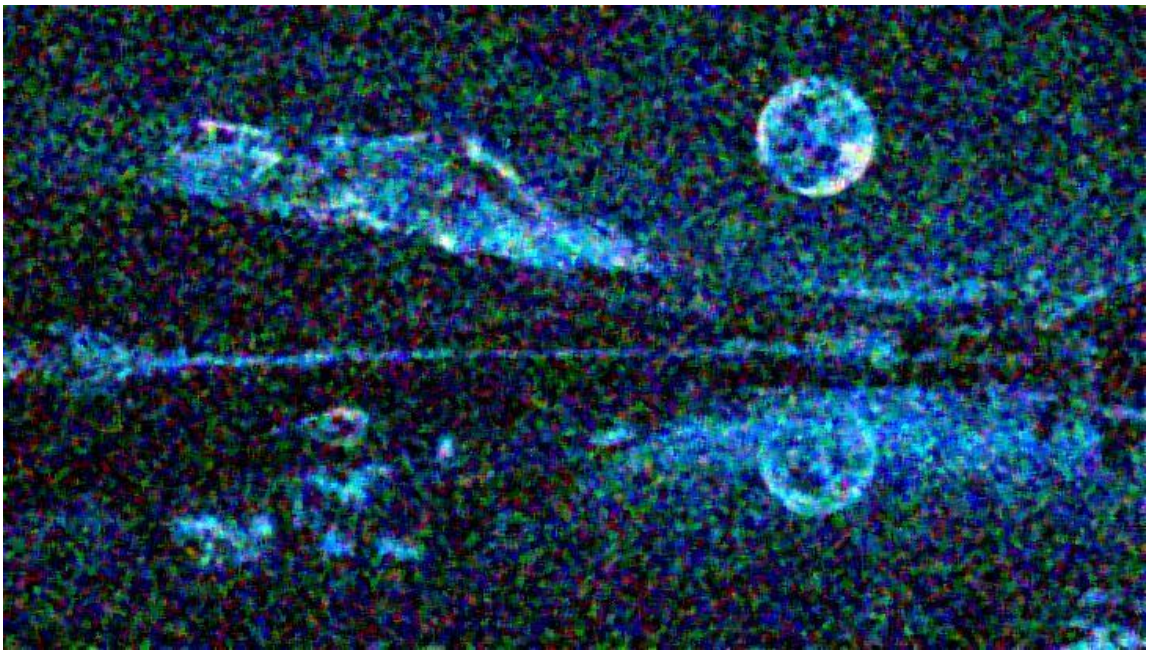


Figure 95 Output of median filter with k = 5: filtered image 1 that was contained high noise (g)

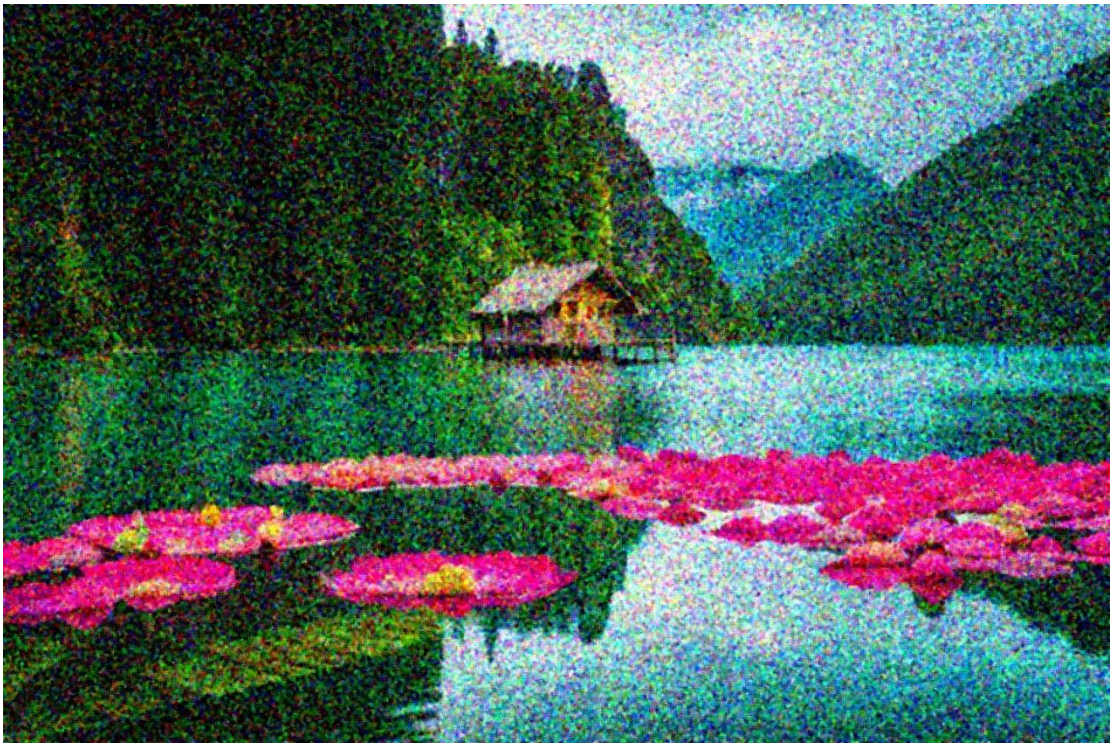


Figure 96 Output of median filter with k = 5: filtered image 1 that was contained high noise (g)

Now, **Kernel Size = 9**

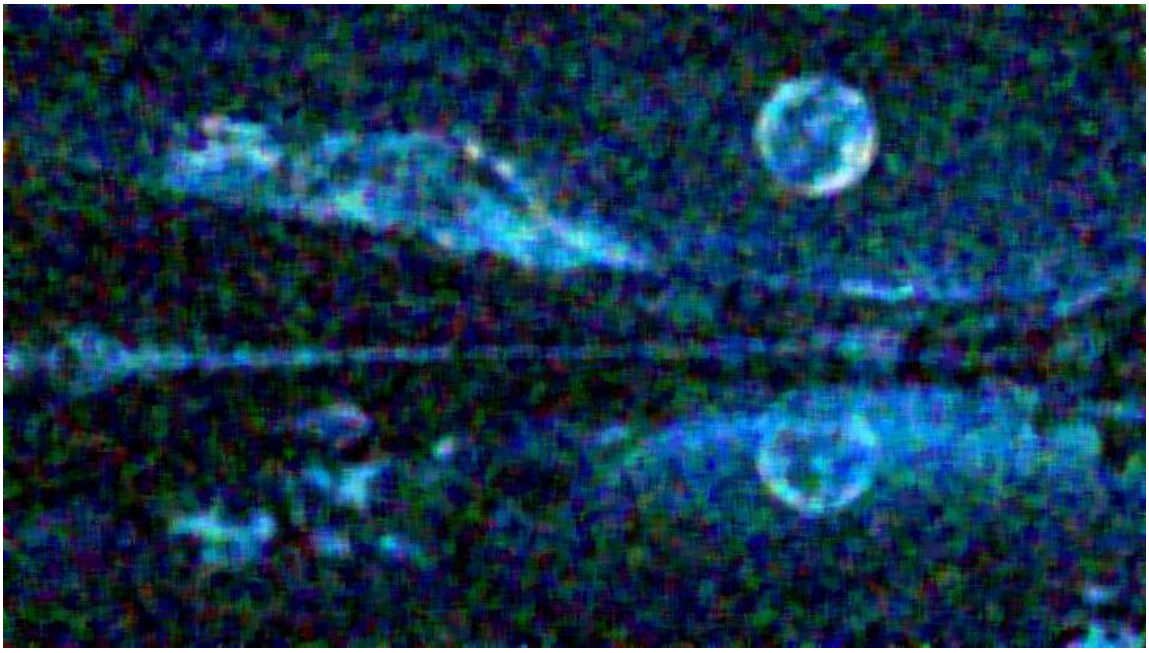


Figure 97 Output of median filter with k = 9: filtered image 1 that was contained high noise (g)

Figure 98 Output of median filter with k = 9: filtered image 1 that was contained high noise (g)

Adaptive Median Filter:

The Adaptive Median filter is a special version of the Median filter that changes the size of the area it looks at based on the image's noise level. Unlike the regular Median filter, which uses a fixed neighborhood size, the Adaptive Median filter starts with a small area and increases it if more noise is detected in that part of the image. This helps it remove salt-and-pepper noise effectively while keeping important details, like edges, intact. The filter adapts to different parts of the image, so it can handle noisy areas better while preserving clean regions. The filter's performance depends on the maximum size of the neighborhood and how many times the filter repeats the process, helping it balance noise removal with edge preservation.

Bilateral Filter:

The Bilateral filter is a special filter that smooths images while preserving edges. It works by considering both the spatial distance between pixels and the intensity difference between them. Pixels that are close to each other in both position and color are blended together more strongly, while pixels that are far apart in either space or color are less affected. This helps the filter

reduce noise while keeping edges sharp, unlike other filters that may blur edges. The Bilateral filter's strength is influenced by the diameter parameter, which controls the size of the neighborhood around each pixel. With the spatial distance and intensity difference parameters fixed at 75, the diameter parameter determines how much surrounding area is considered when filtering each pixel. A larger diameter allows the filter to consider a wider area, leading to stronger smoothing, while a smaller diameter focuses more on nearby pixels, helping to preserve edges more effectively. By adjusting the diameter, the filter can balance noise reduction with edge preservation.

Now, we will apply this filter to both noisy images at low, medium, and high noise levels. For each noise level, we will vary the diameter to observe the changes in the filtered images as follows.

Salt and Pepper Low Noise diameter= 9

Figure 99: Output of bilateral filter with d = 9: filtered image 1 that was contained low noise (s & p)

Figure 100: Output of bilateral filter with d = 9: filtered image 2 that was contained low noise (s & p)

Now, **diameter = 15**

Figure 101: Output of bilateral filter with d = 15: filtered image 1 that was contained low noise (s & p)

Figure 102: Output of bilateral filter with d = 15: filtered image 2 that was contained low noise (s & p)

Salt and Pepper Medium Noise and diameter = 9

Figure 103: Output of bilateral filter with d = 9: filtered image 1 that was contained medium noise (s & p)

Figure 104: Output of bilateral filter with d = 9: filtered image 2 that was contained medium noise (s & p)

Now, **diameter = 15**

Figure 105: Output of bilateral filter with d = 15: filtered image 1 that was contained medium noise (s & p)

Figure 106: Output of bilateral filter with d = 15: filtered image 2 that was contained medium noise (s & p)

Salt and Pepper High Noise and diameter = 9

Figure 107: Output of bilateral filter with d = 9: filtered image 1 that was contained high noise (s & p)

Figure 108: Output of bilateral filter with d = 9: filtered image 2 that was contained high noise (s & p)

Now, **diameter = 15**

Figure 109: Output of bilateral filter with d = 15: filtered image 1 that was contained high noise (s & p)

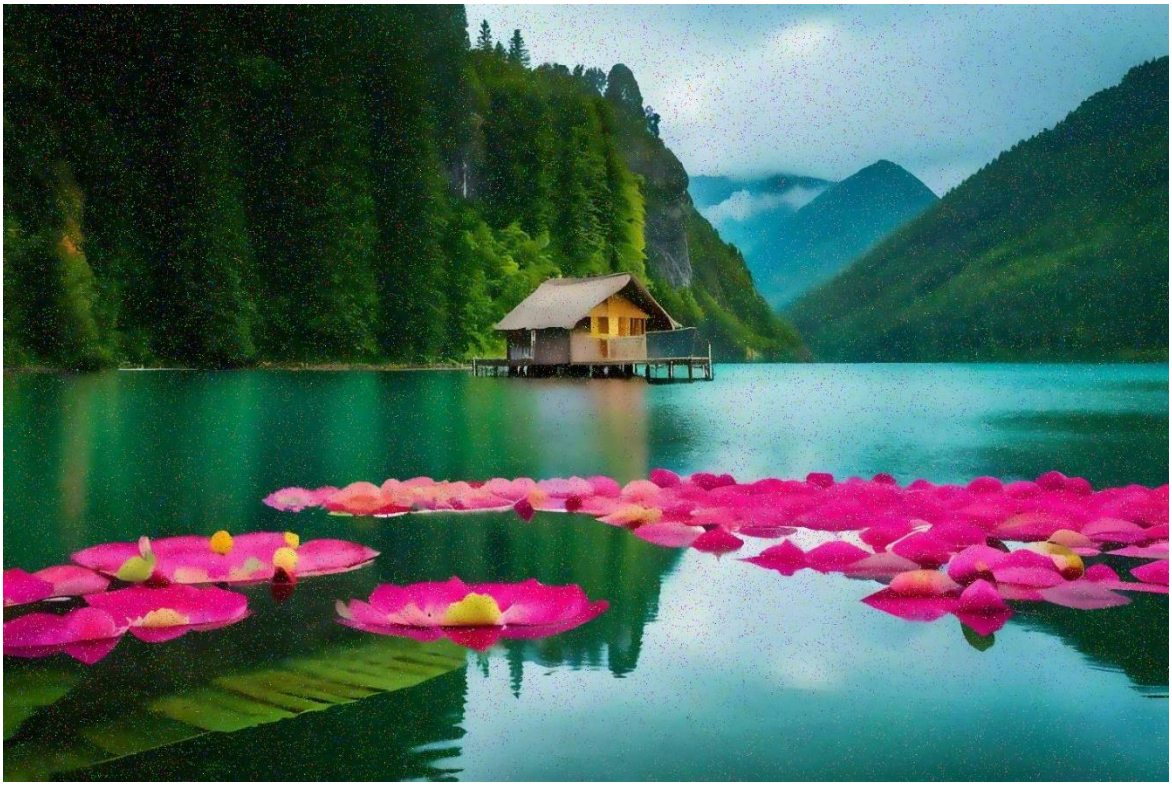


Figure 110: Output of bilateral filter with d = 15: filtered image 2 that was contained high noise (s & p)

Gaussian Low Noise diameter= 9

Figure 111: Output of bilateral filter with d = 9: filtered image 1 that was contained low noise (g)

Figure 112: Output of bilateral filter with d = 9: filtered image 2 that was contained low noise (g)

Now, **diameter = 15**

Figure 113: Output of bilateral filter with d = 15: filtered image 1 that was contained low noise (g)

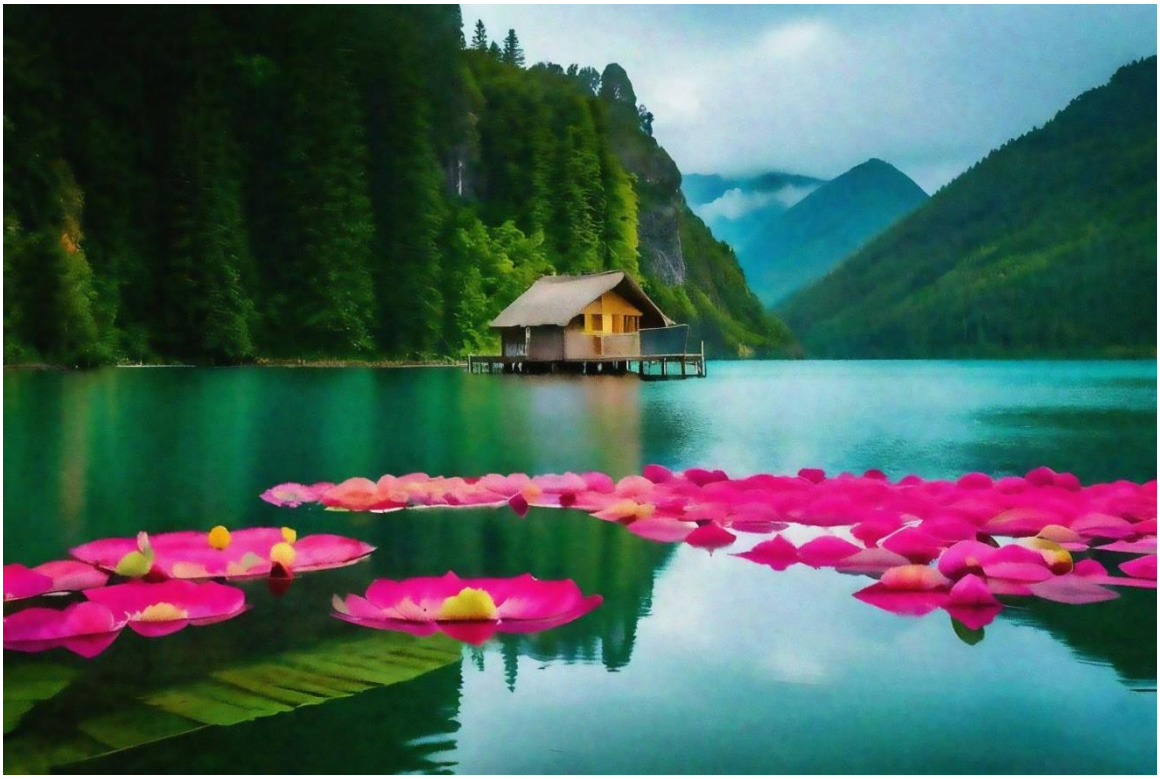


Figure 114: Output of bilateral filter with d = 15: filtered image 2 that was contained low noise (g)

Gaussian Medium Noise and diameter = 9

Figure 115: Output of bilateral filter with d = 9: filtered image 1 that was contained medium noise (g)

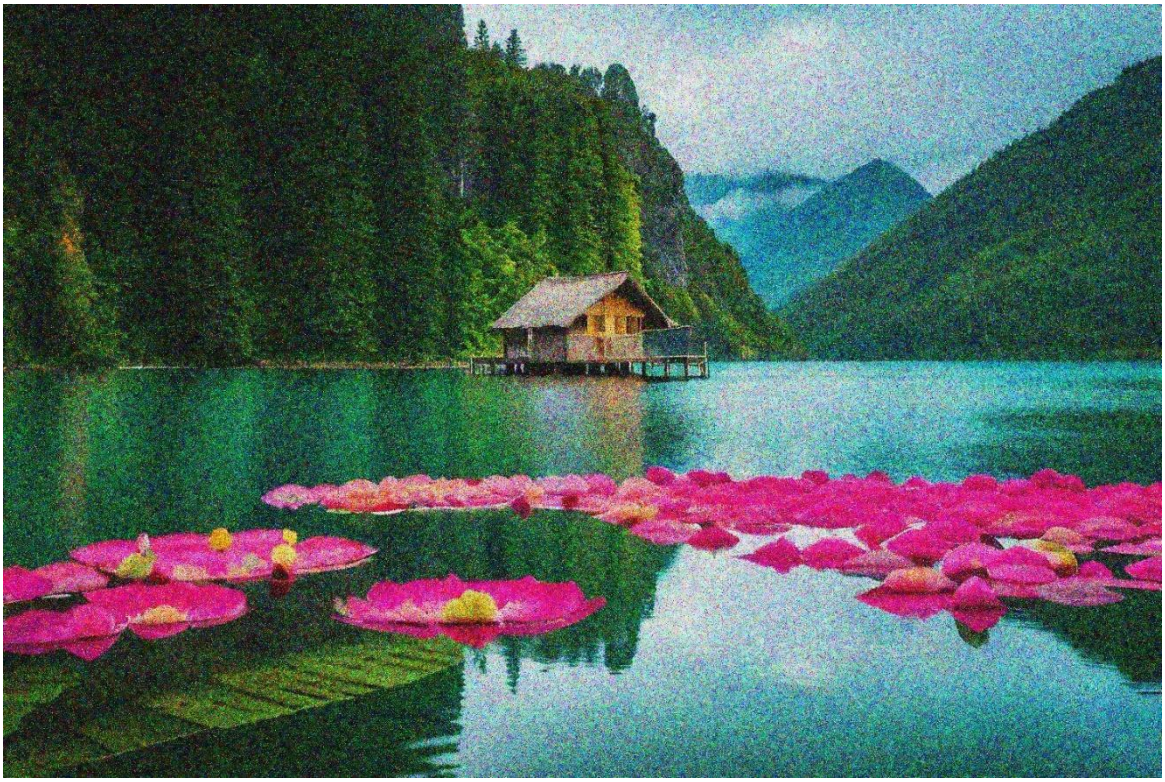


Figure 116: Output of bilateral filter with d = 9: filtered image 2 that was contained medium noise (g)

Now, **diameter = 15**

Figure 117: Output of bilateral filter with d = 15: filtered image 1 that was contained medium noise (g)

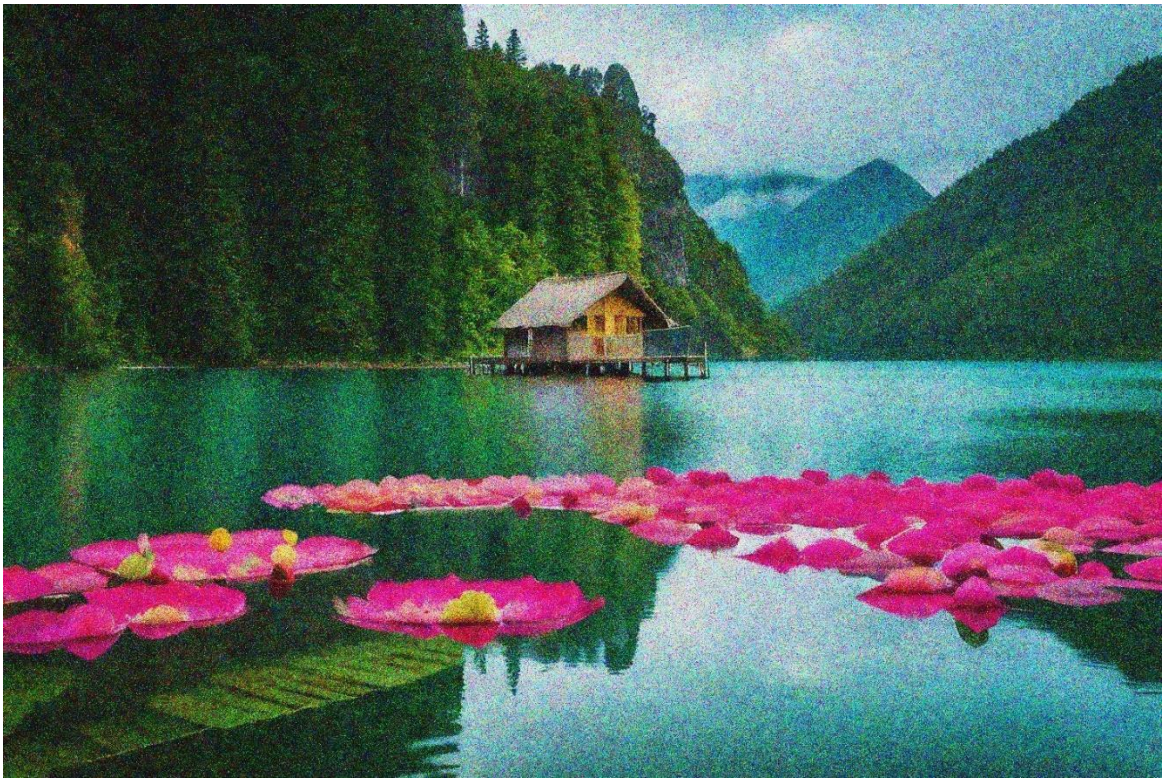


Figure 118: Output of bilateral filter with d = 15: filtered image 2 that was contained medium noise (g)

Gaussian High Noise and diameter = 9

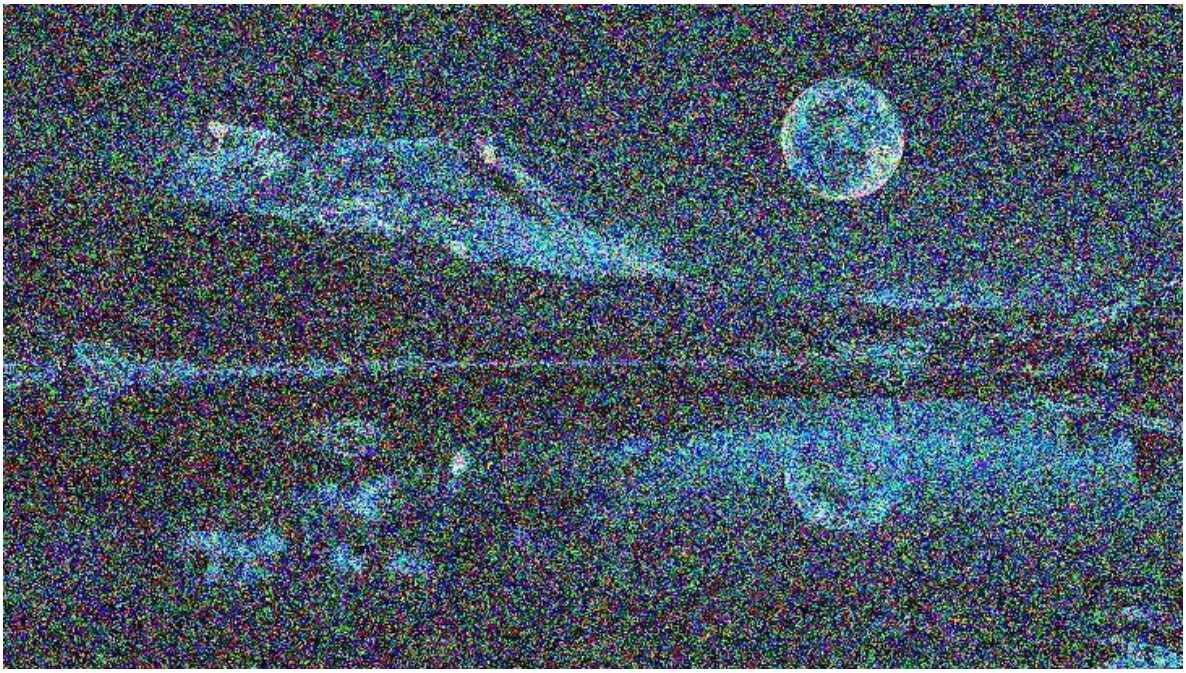


Figure 119: Output of bilateral filter with d = 9: filtered image 1 that was contained high noise (g)

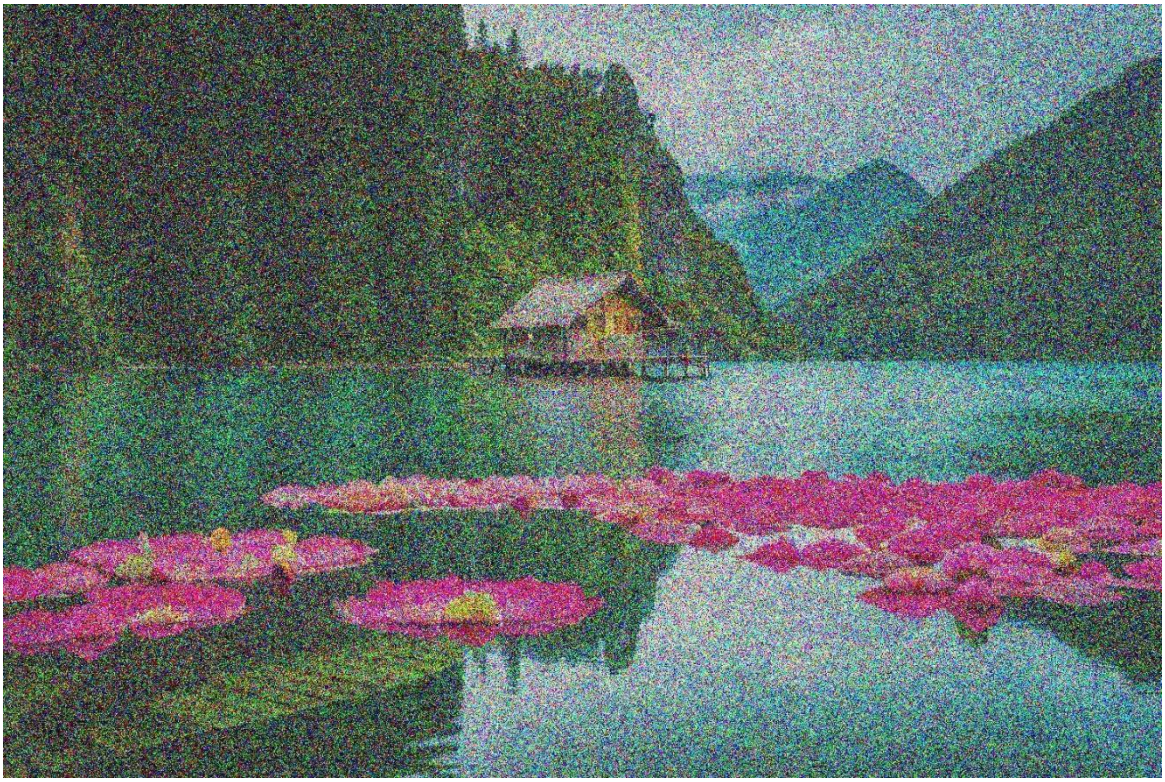


Figure 120: Output of bilateral filter with d = 9: filtered image 2 that was contained high noise (g)

Now, **diameter = 15**

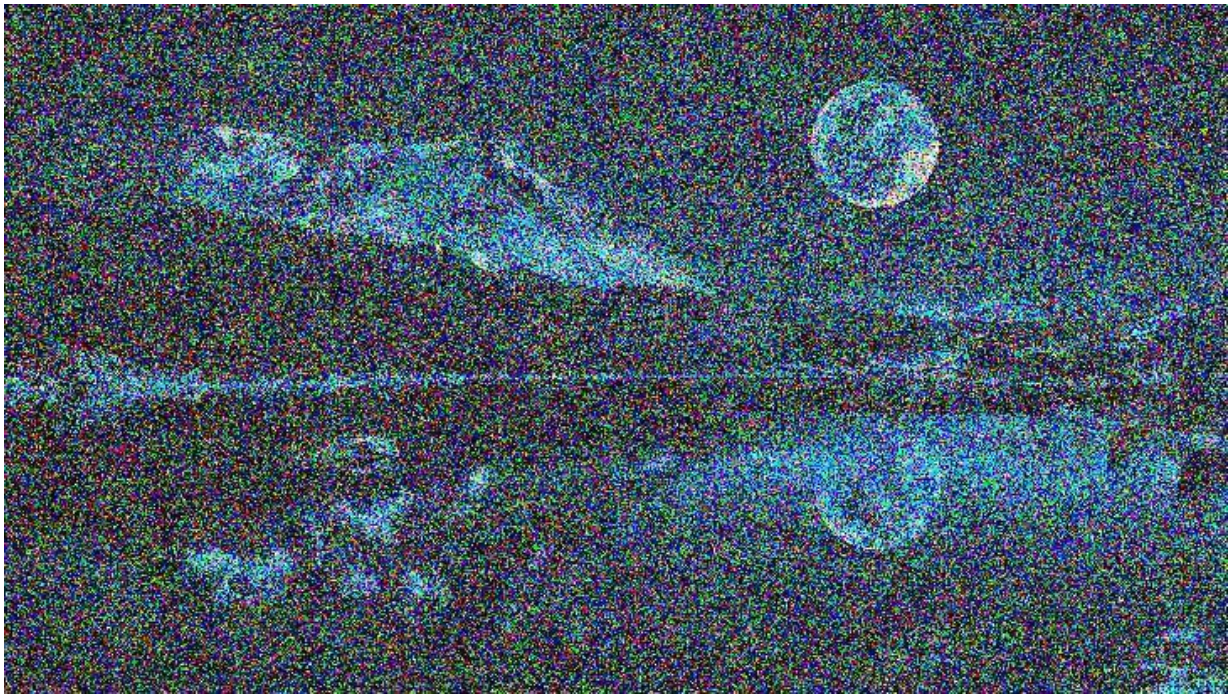


Figure 121: Output of bilateral filter with d = 15: filtered image 1 that was contained high noise (g)

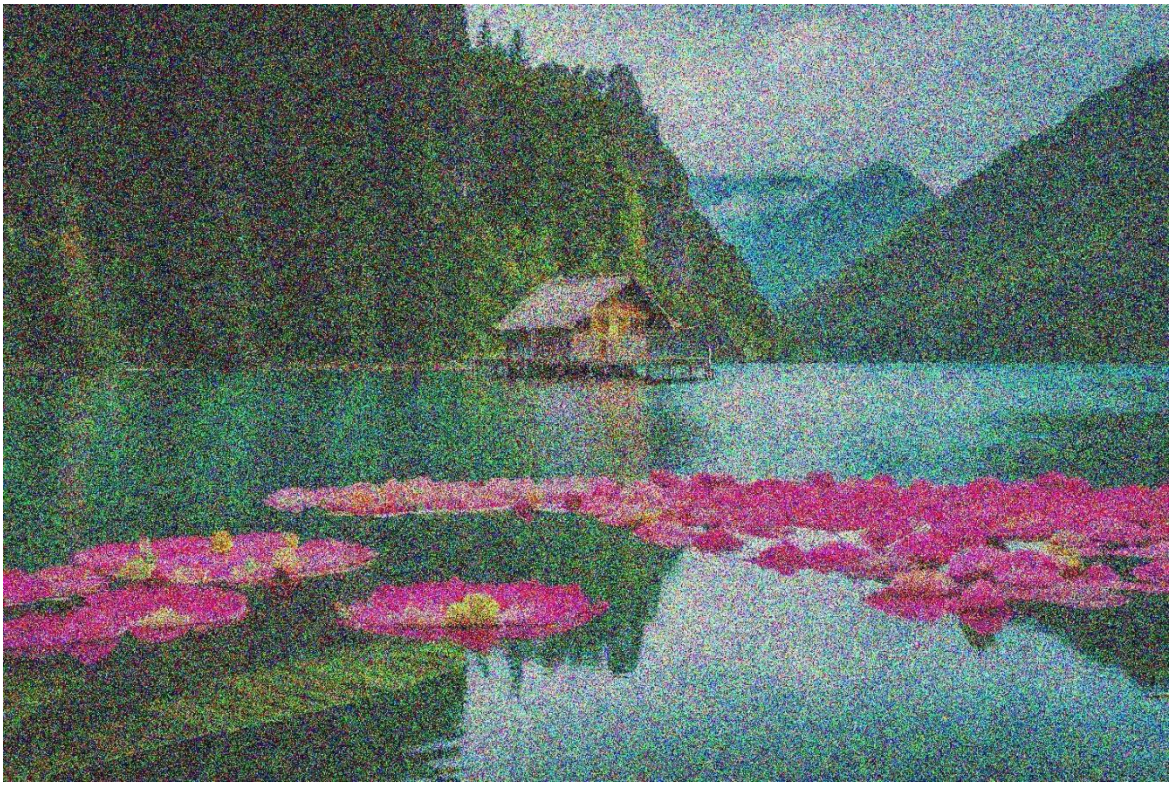


Figure 122: Output of bilateral filter with d = 15: filtered image 2 that was contained high noise (g)

All Filtered Images with kernel = 3 for Basic Filters and diameter =9 for Bilateral Filter

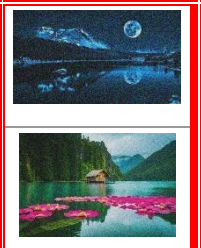
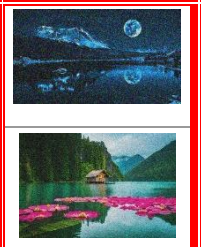
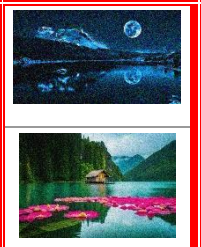
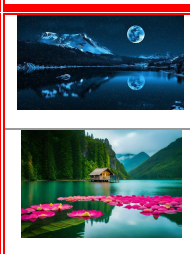
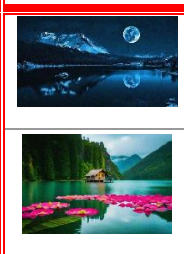
Filter Type/Noise	Low S&P	Medium S&P	High S&P	Low G	Medium G	High G
Box Filter						
Gaussian Filter						
Median Filter						
Bilateral Filter						

Table 0-1: All Filtered Images through various filters

Step 3: Measuring Performance

When Salt and Pepper Low noise applied on first image for example, the MSE, PSNR and Computational Time measurements executed using skimage. metrics library as following in table below:

Metric\Filter		Box	Gaussian	Median	Bilateral	
MSE	K = 3	147.33	110.81	104.99	179.50	D=9
	K = 5	218.27	139.12	186.91	143.57	D=15
	K = 9	310.02	202.87	291.63	131.71	D=29
PSNR	K = 3	26.45	27.69	24.92	25.59	D=9
	K = 5	24.74	47.17	25.41	26.56	D=15
	K = 9	23.22	25.06	23.48	26.93	D=29
Computational Time	K = 3	1 ms	1 ms	1 ms	21 ms	D=9
	K = 5	1 ms	2 ms	2 ms	58 ms	D=15
	K = 9	1 ms	2 ms	20 ms	210 ms	D=29

Table 0-2: Metrics Measurements

Applying Canny Edge Detector:

- **Box Filter**

Salt and Pepper Low Noise with **K = 3, K = 5 and K = 9** Respectively:

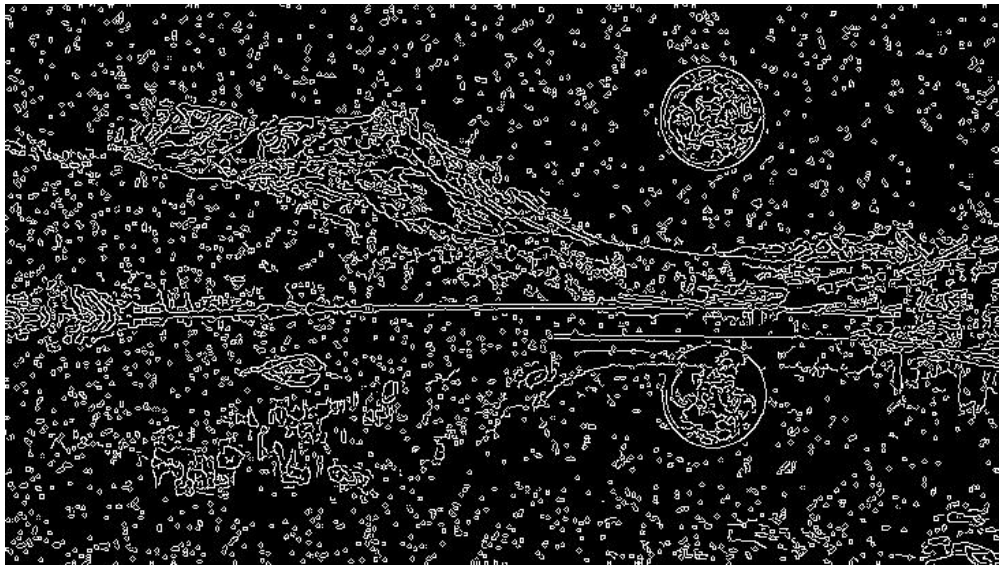


Figure 123: Output of Box Filter after applying Canny at K = 3 of image 1

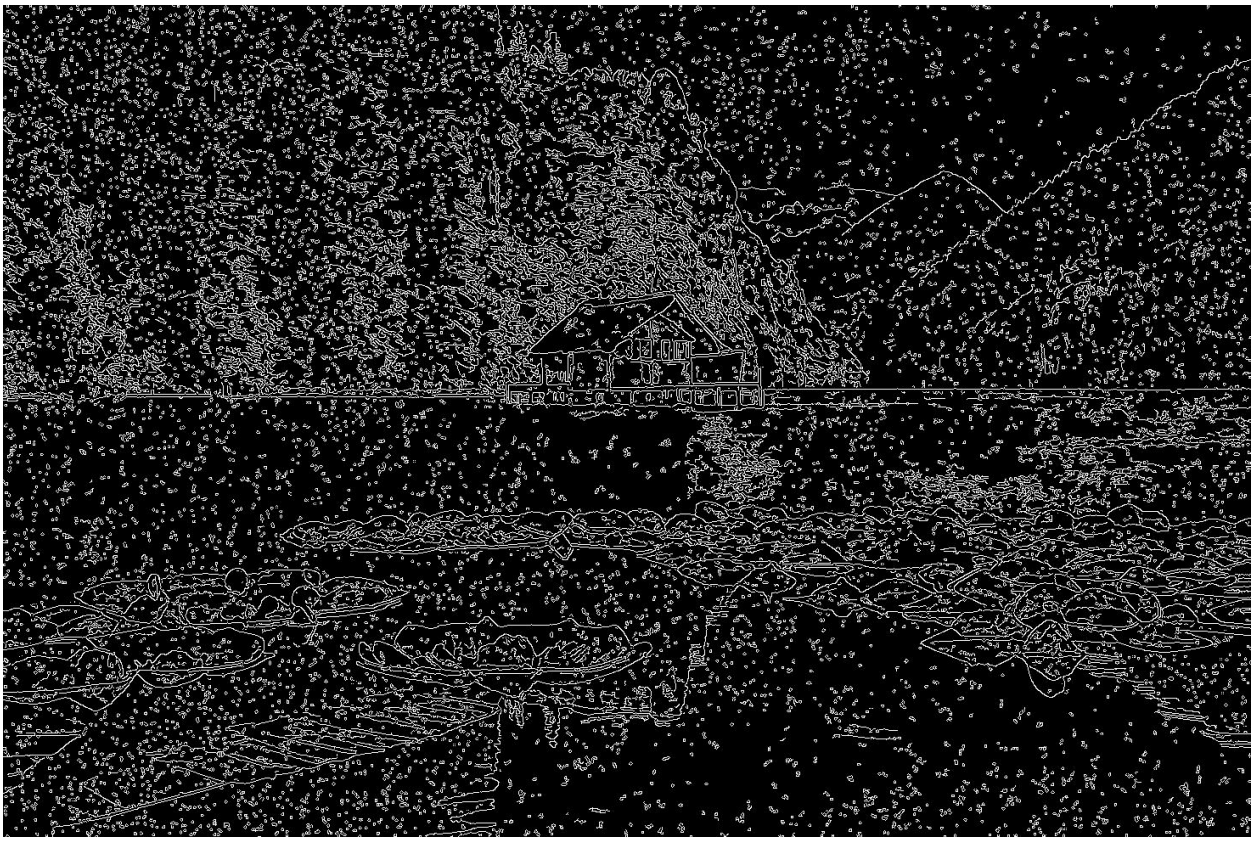


Figure 124: Output of Box Filter after applying Canny at K = 3 of image 2

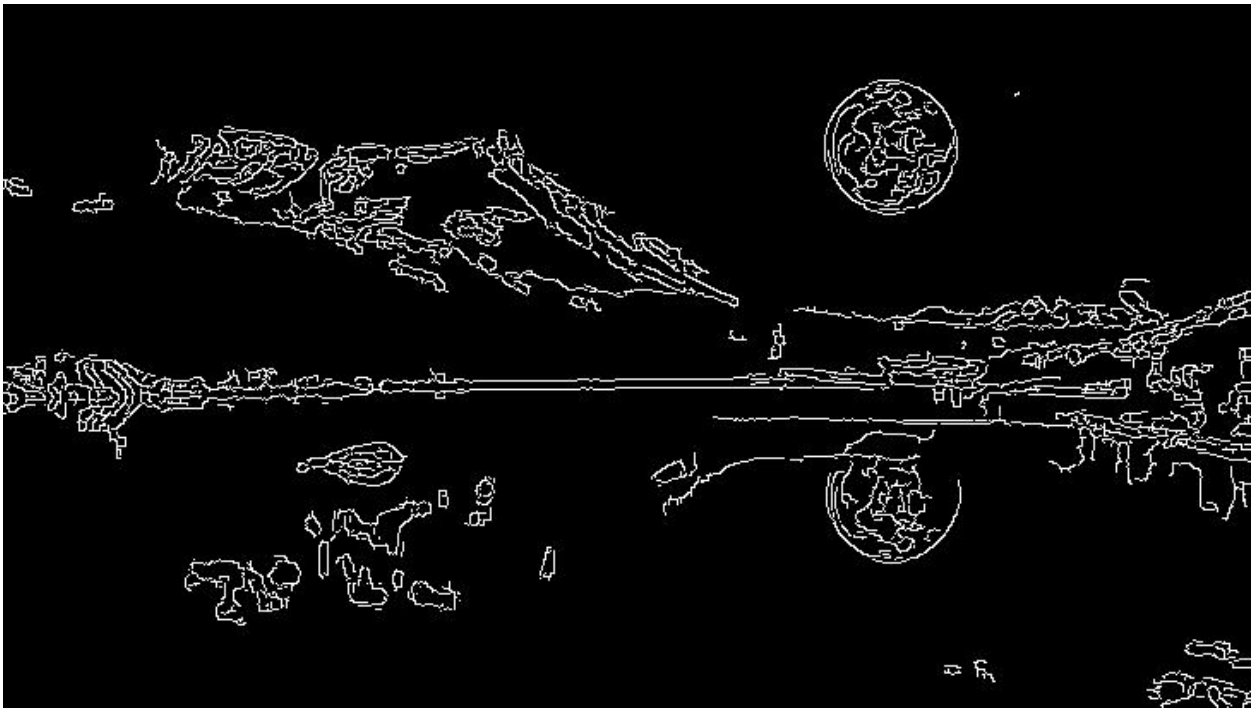


Figure 125: Output of Box Filter after applying Canny at K = 5 of image 1

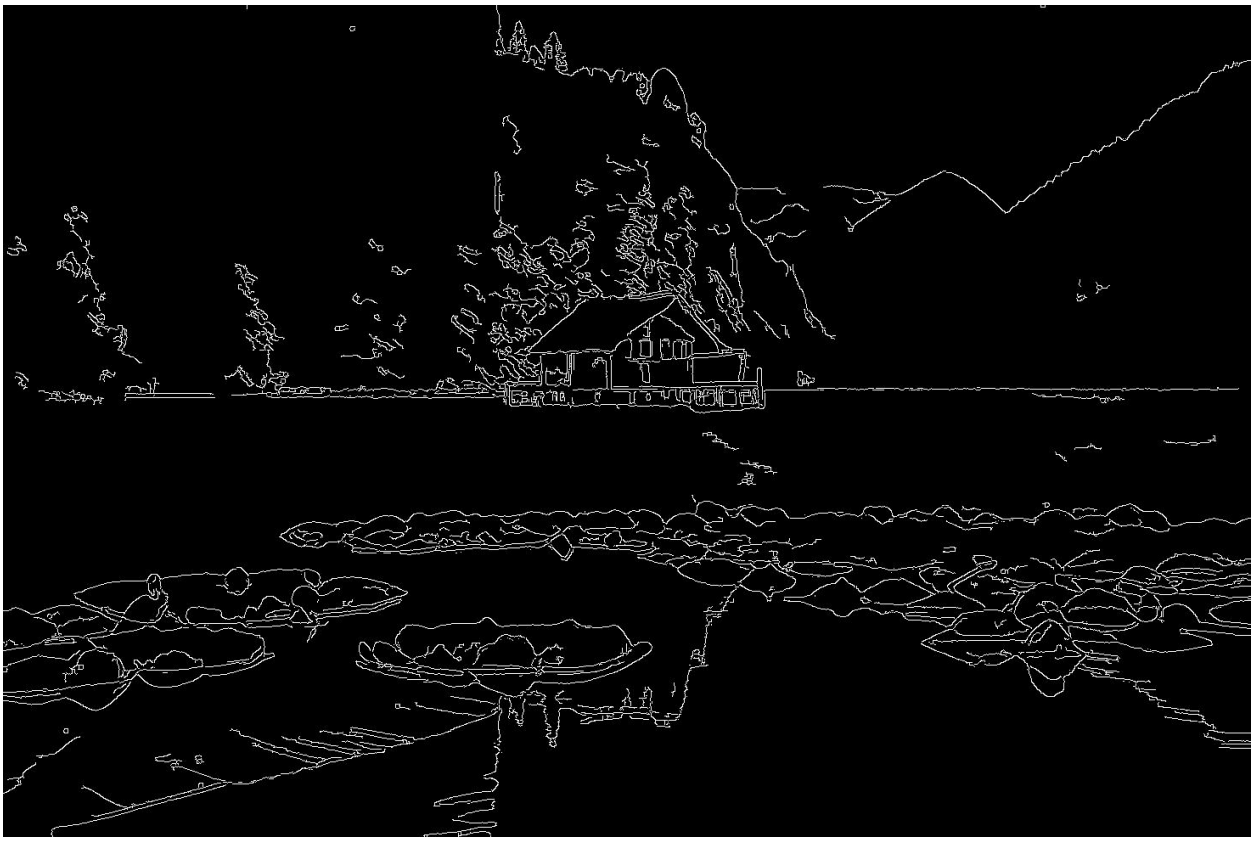


Figure 126: Output of Box Filter after applying Canny at K = 5 of image 2

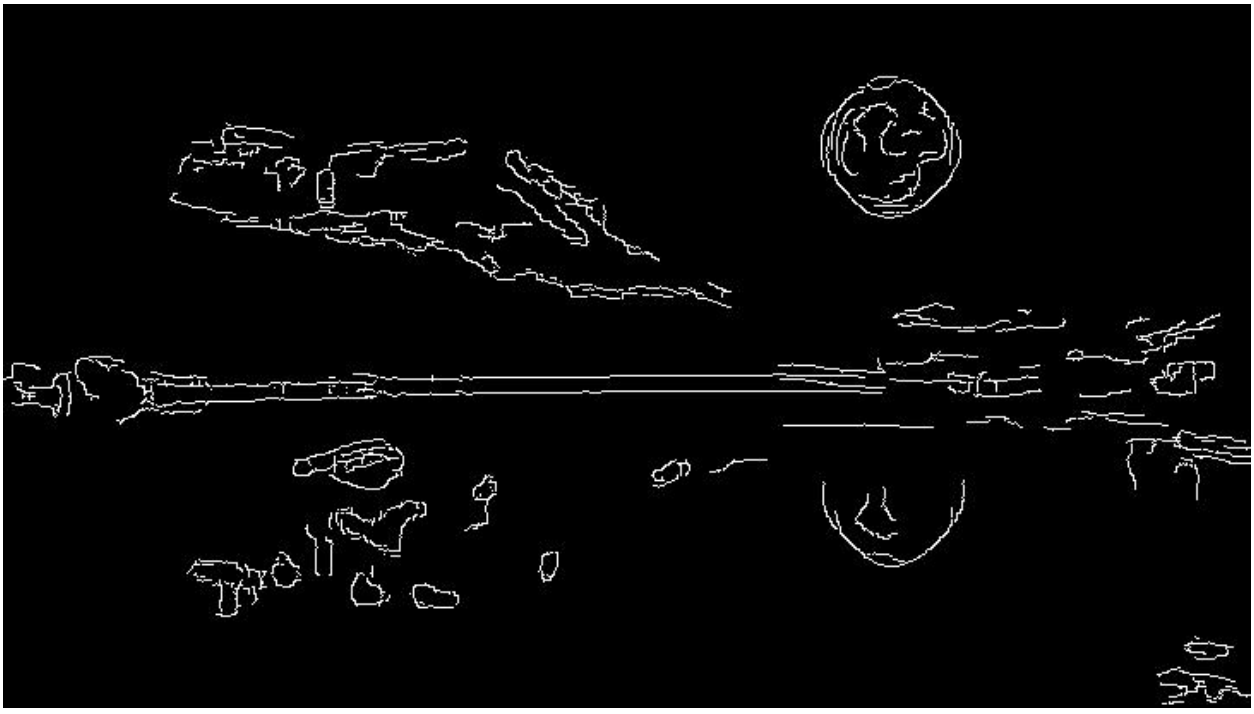


Figure 127: Output of Box Filter after applying Canny at K = 9 of image 1

Figure 128: Output of Box Filter after applying Canny at K = 9 of image 2

- **Gaussian Filter**

Salt and Pepper Low Noise with **K = 3, K = 5 and K = 9** Respectively:

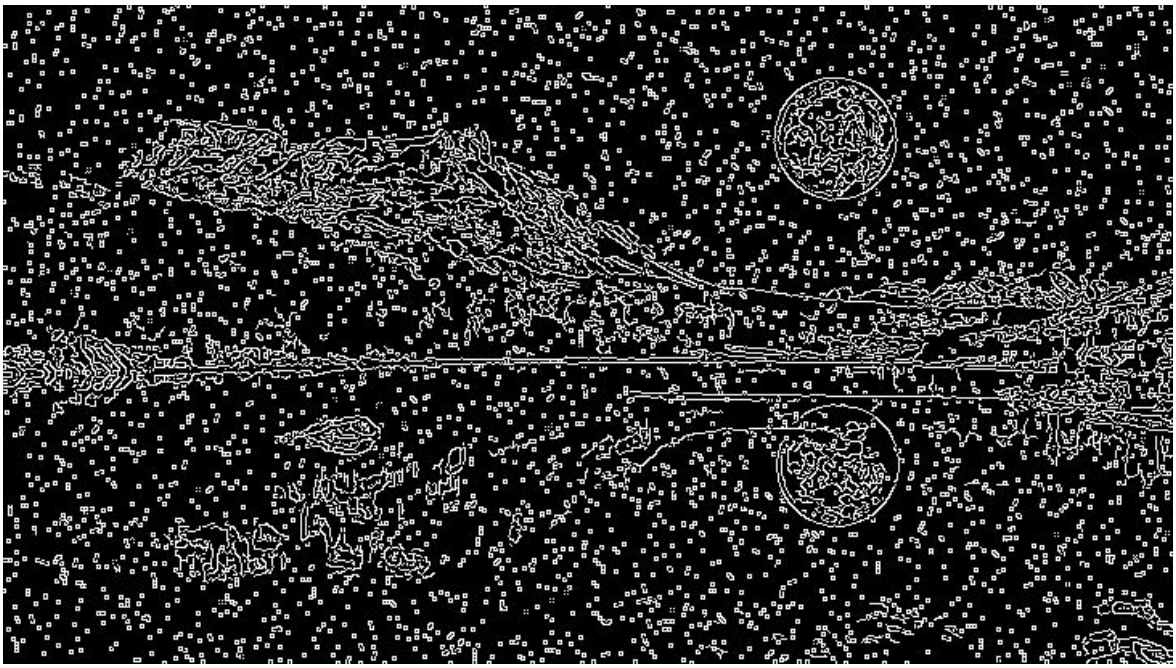


Figure 129: Output of Gaussian Filter after applying Canny at K = 3 of image 1

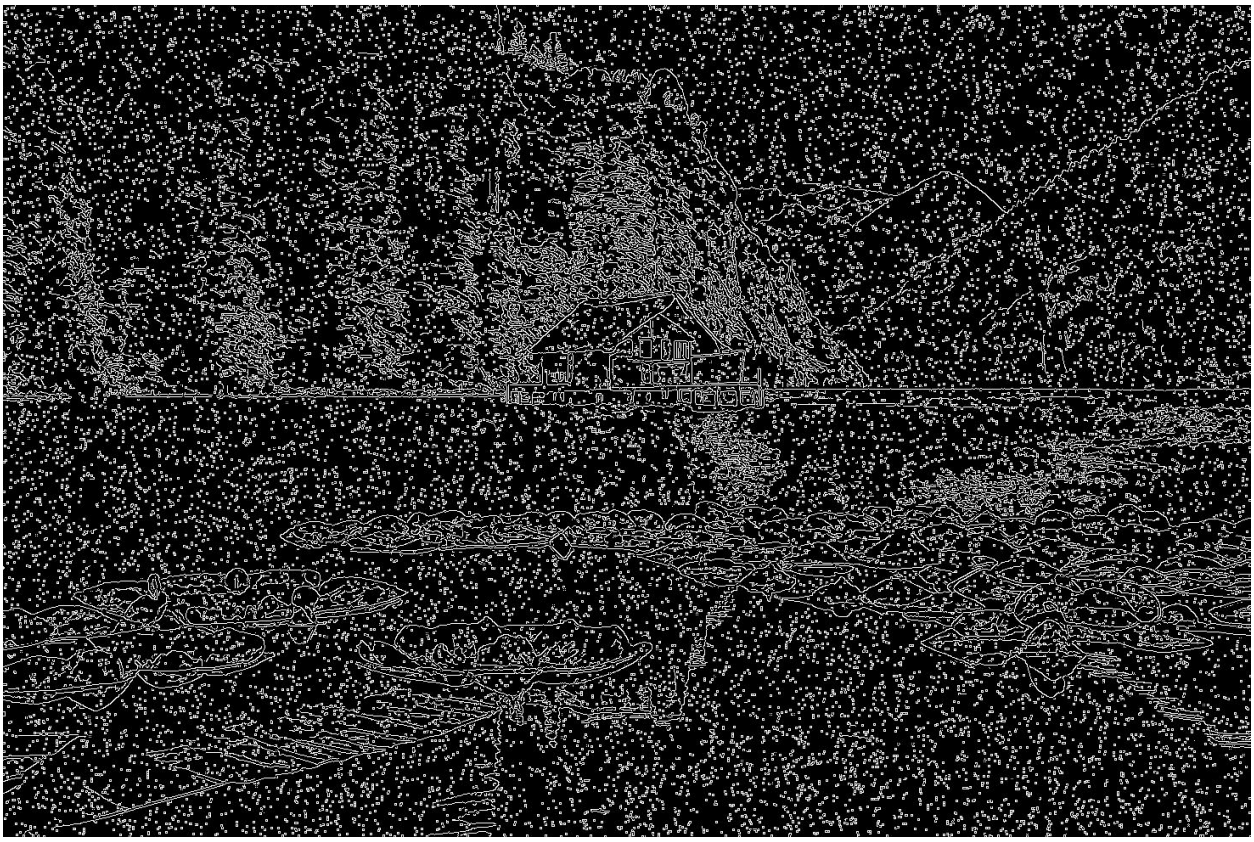


Figure 130: Output of Gaussian Filter after applying Canny at K = 3 of image 2

Figure 131: Output of Gaussian Filter after applying Canny at K = 5 of image 1

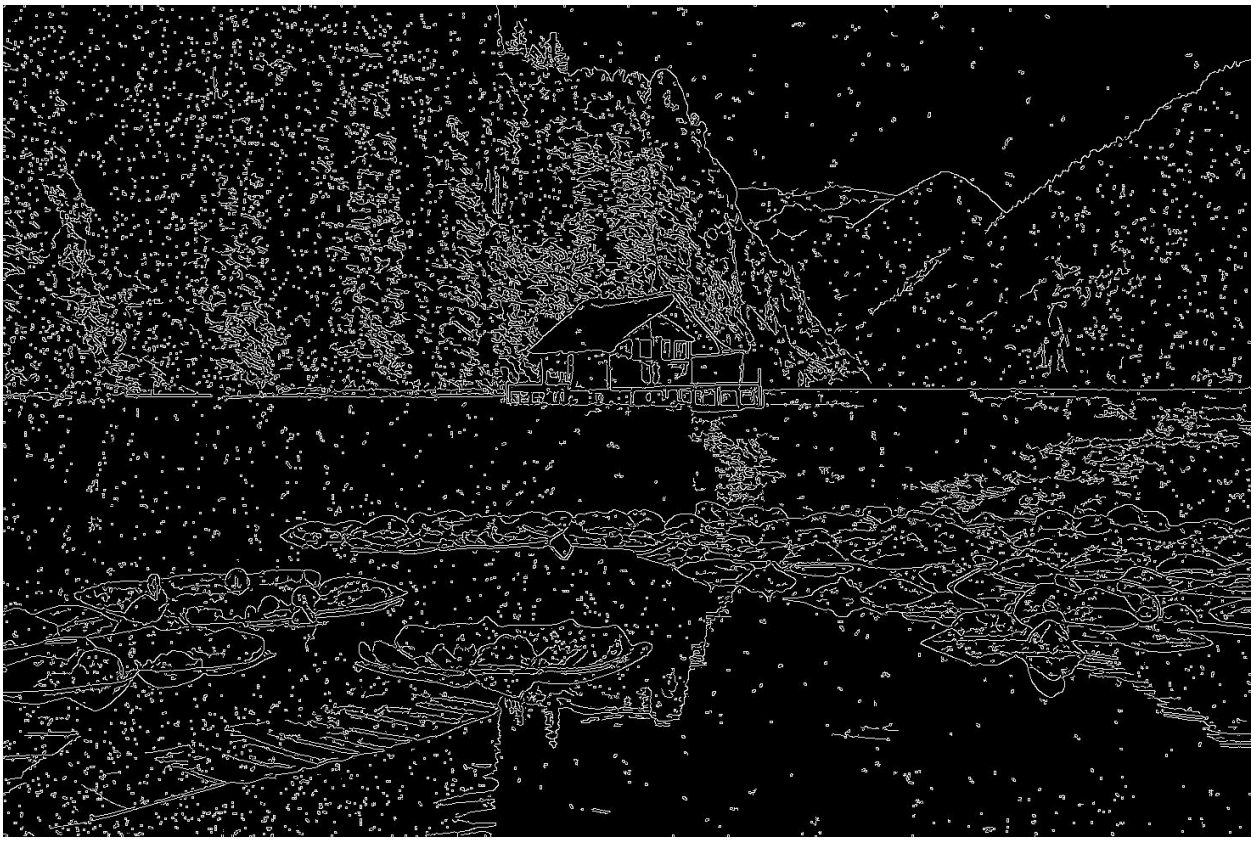


Figure 132: Output of Gaussian Filter after applying Canny at K = 5 of image 2

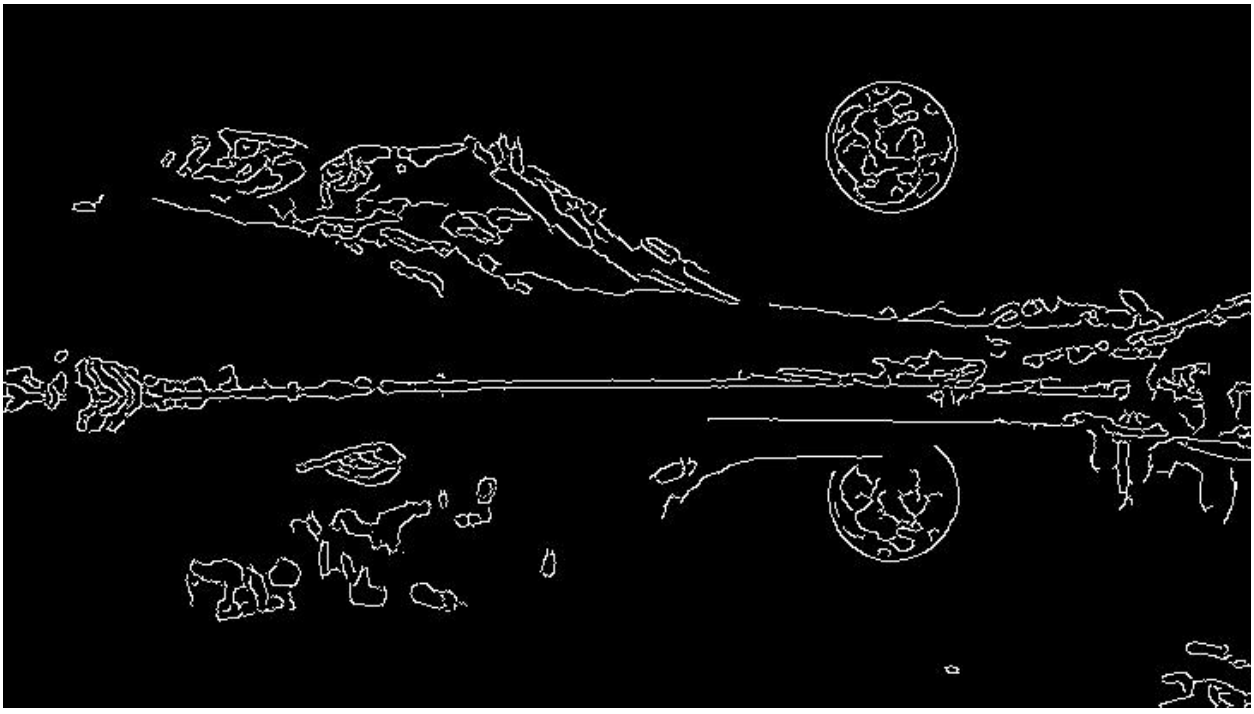


Figure 133: Output of Gaussian Filter after applying Canny at K = 9 of image 1

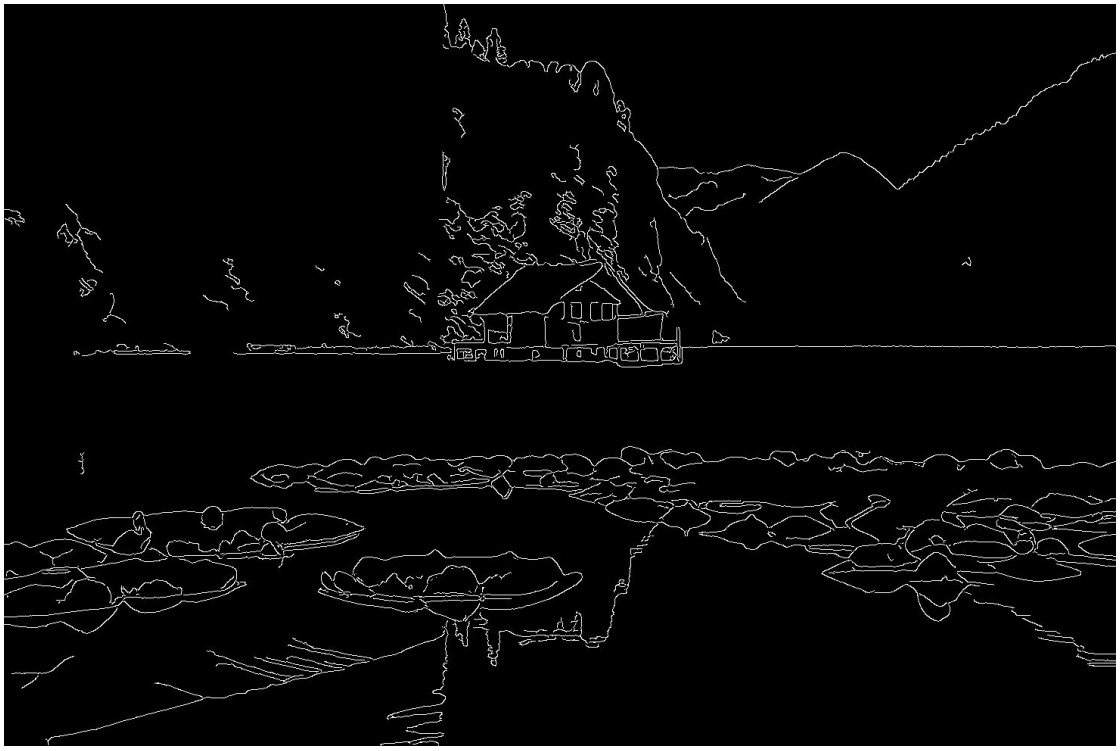


Figure 134: Output of Gaussian Filter after applying Canny at K = 9 of image 2

- **Median Filter**

Salt and Pepper Low Noise with **K = 3, K = 5 and K = 9** Respectively:

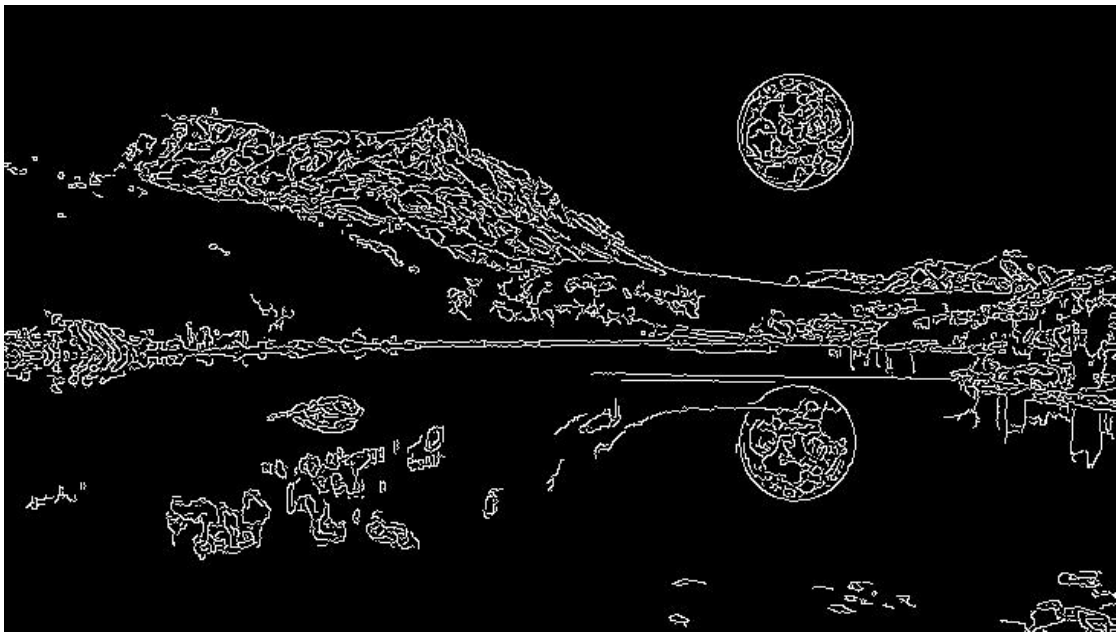


Figure 135: Output of Median Filter after applying Canny at K = 3 of image 1

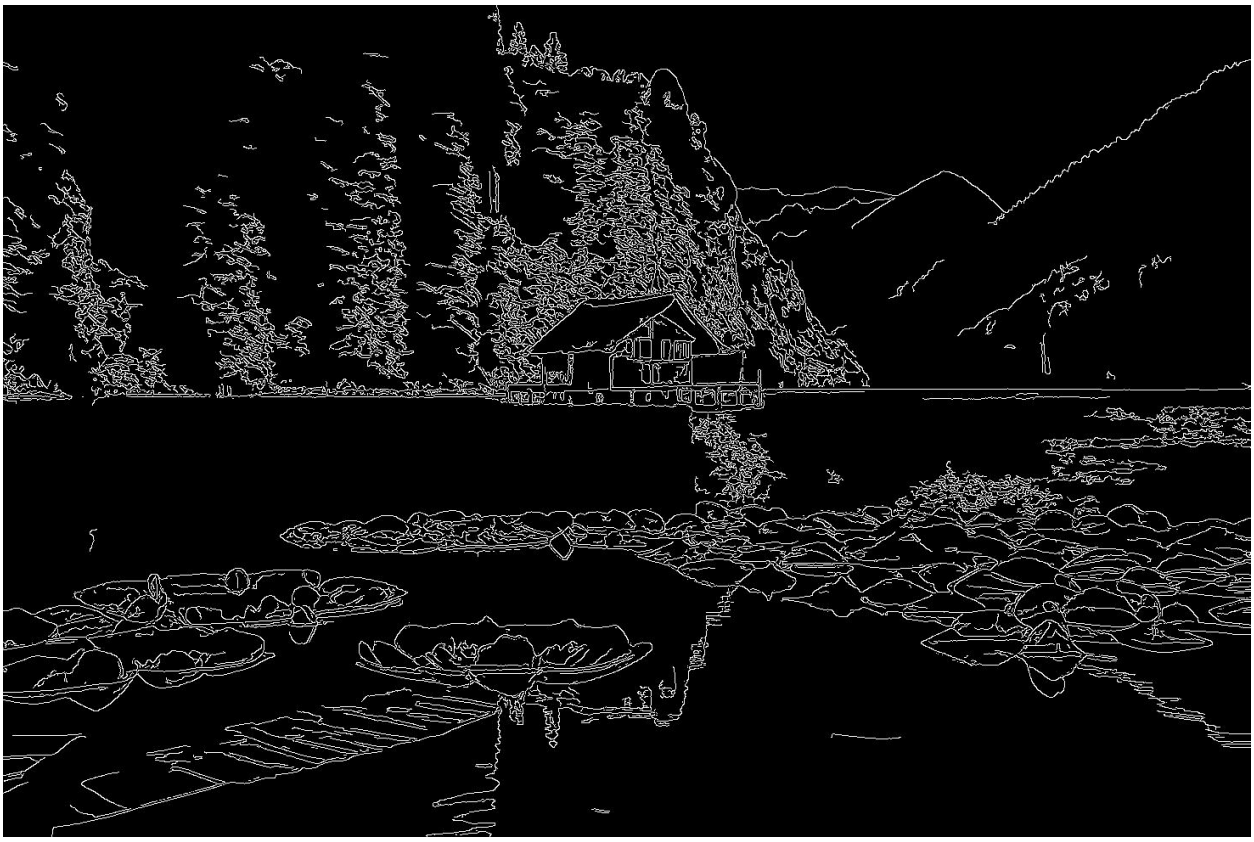


Figure 136: Output of Median Filter after applying Canny at K = 3 of image 2

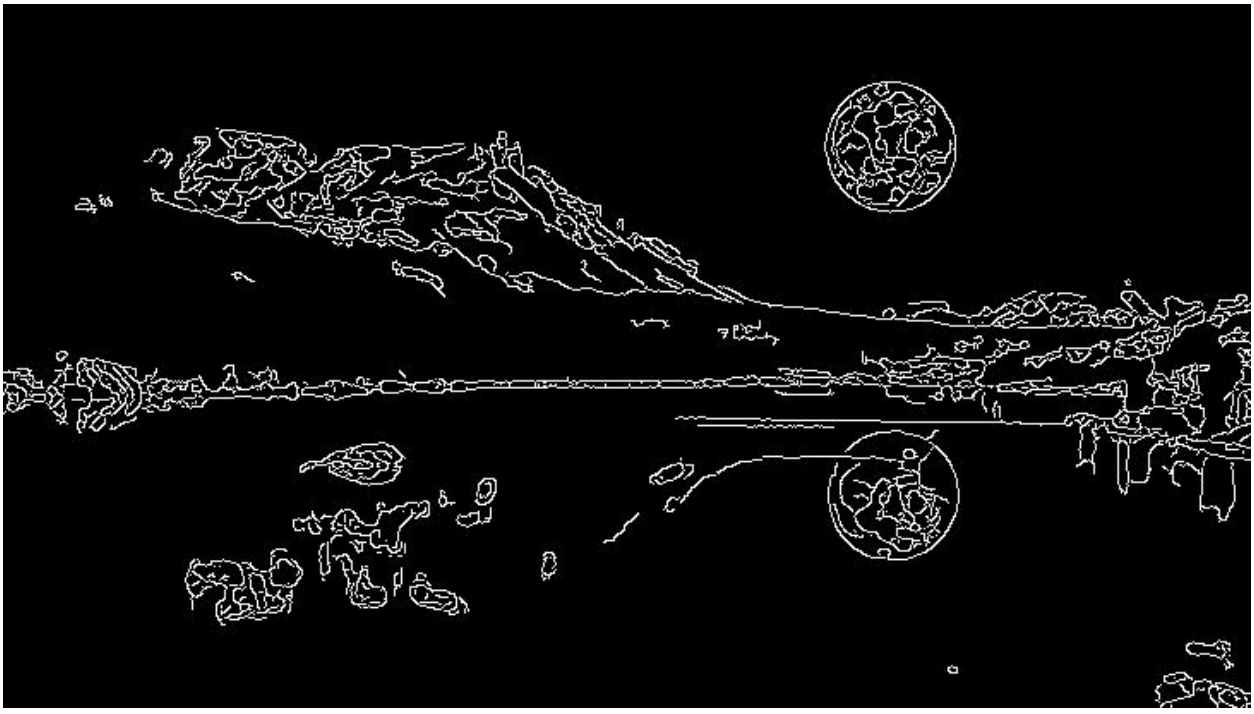


Figure 137: Output of Median Filter after applying Canny at K = 5 of image 1

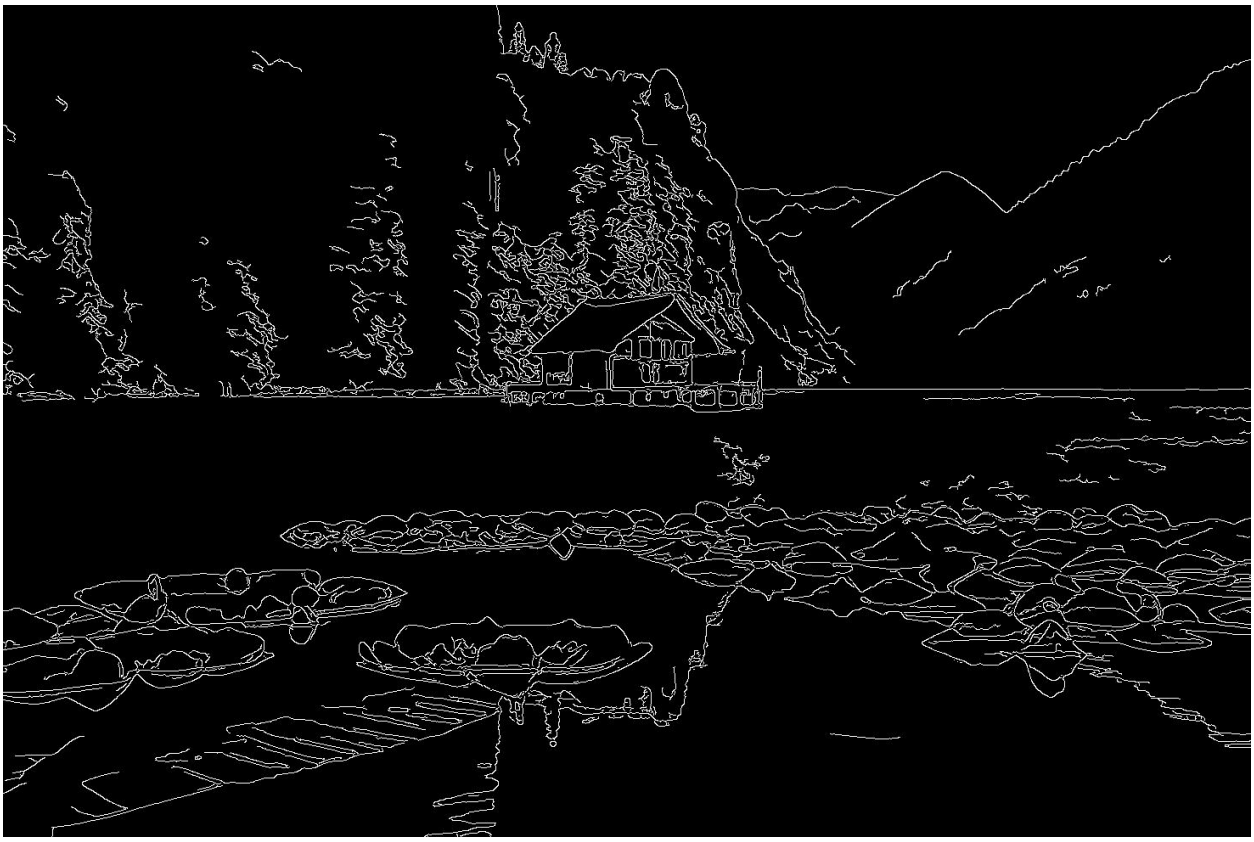


Figure 138: Output of Median Filter after applying Canny at K = 5 of image 2

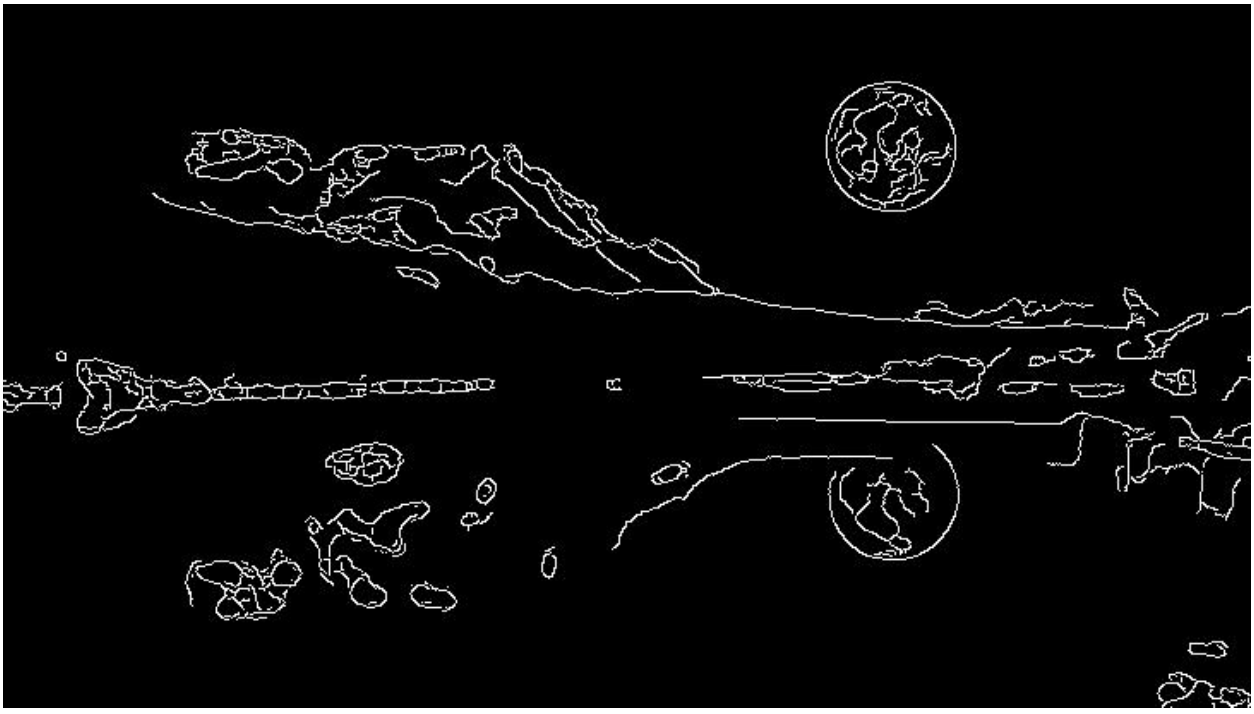


Figure 139: Output of Median Filter after applying Canny at K = 9 of image 1

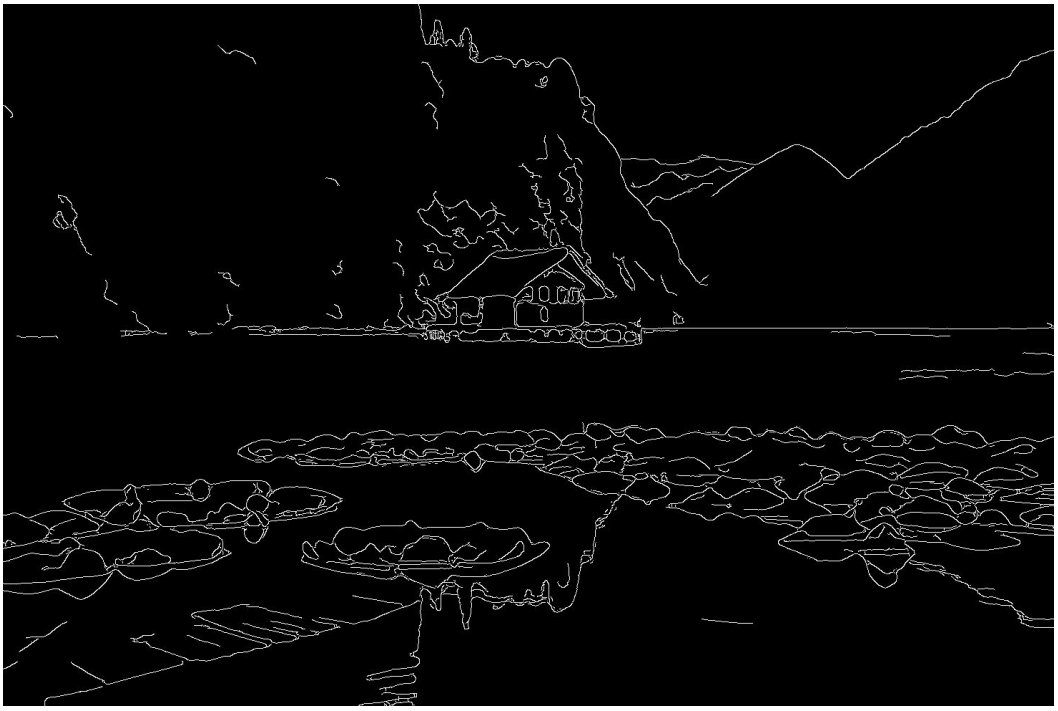


Figure 140: Output of Median Filter after applying Canny at K = 9 of image 2

- **Bilateral Filter**

Salt and Pepper Low Noise with **D = 9, D = 15 and D = 29** Respectively:

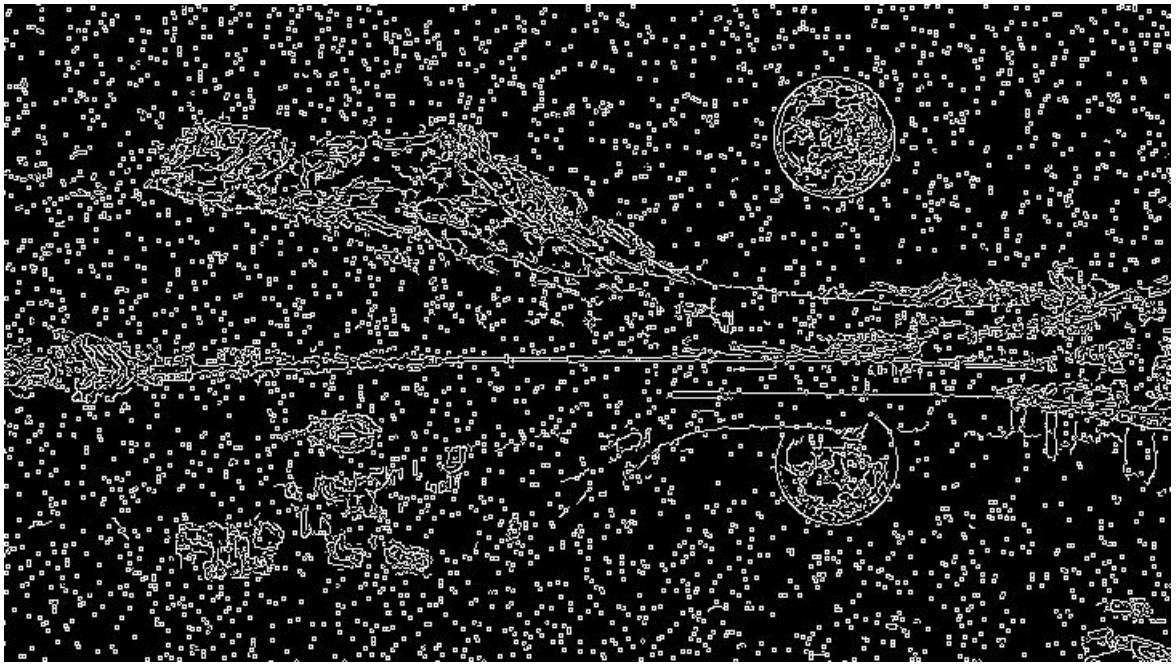


Figure 141: Output of Bilateral Filter after applying Canny at D = 9 of image 1

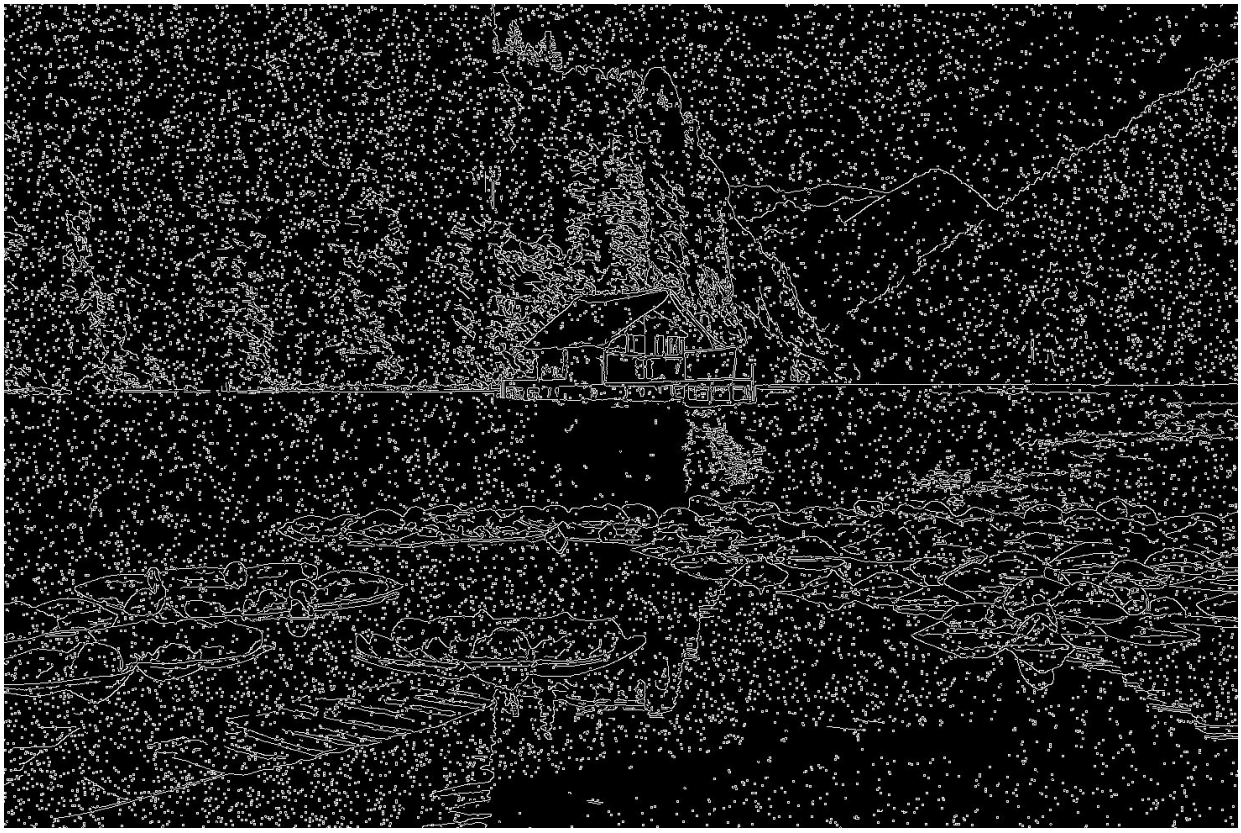


Figure 142: Output of Bilateral Filter after applying Canny at D = 9 of image 2

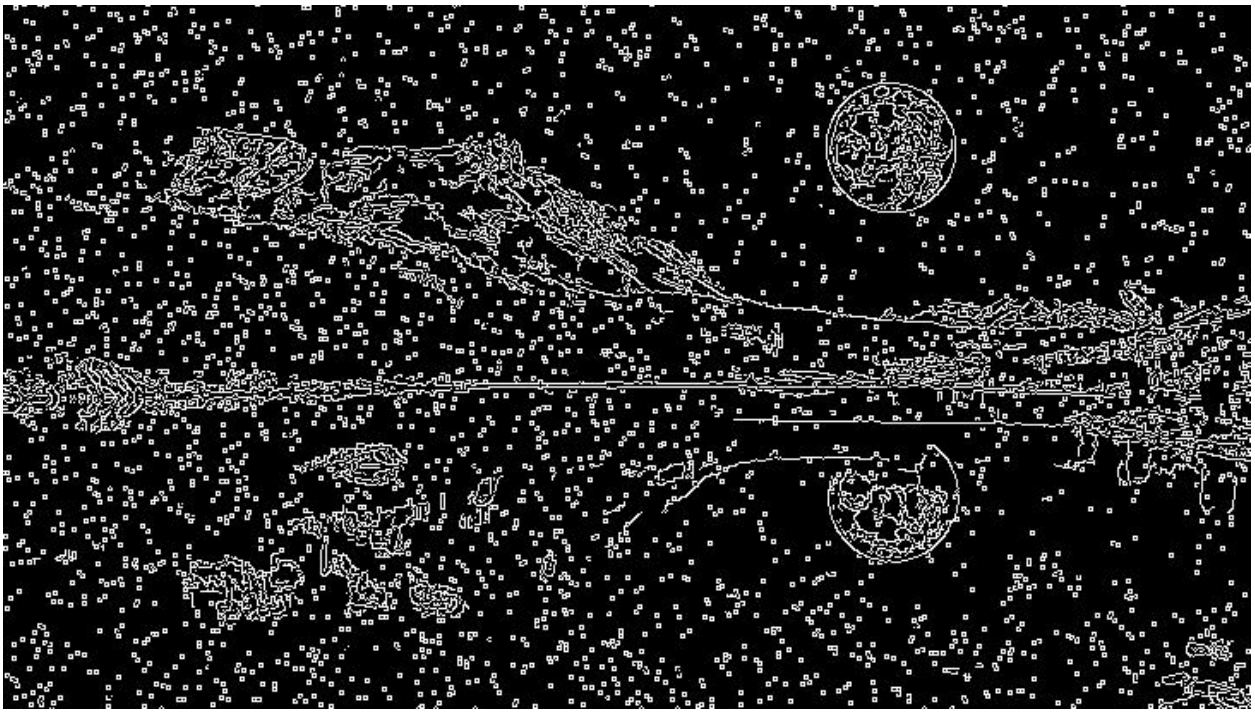


Figure 143: Output of Bilateral Filter after applying Canny at D = 15 of image 1

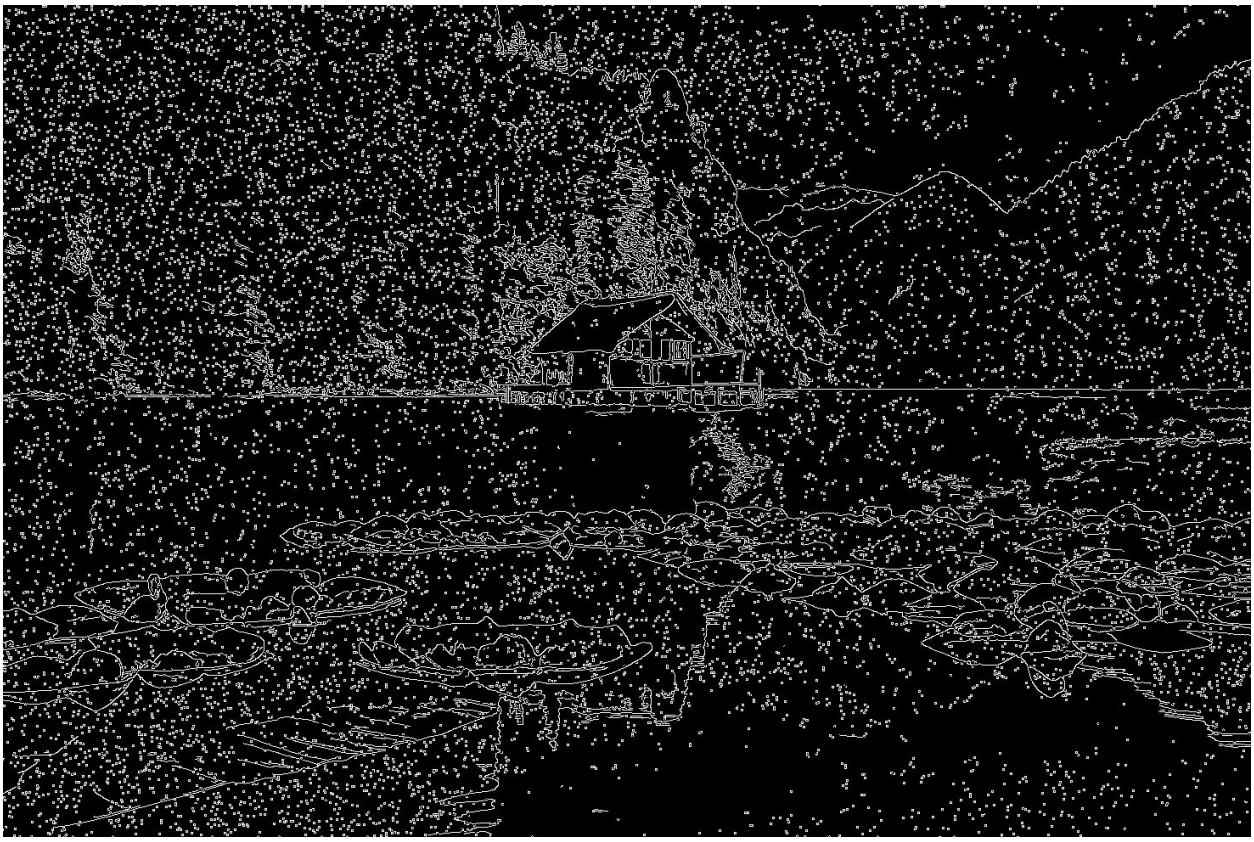


Figure 144: Output of Bilateral Filter after applying Canny at D = 15 of image 2

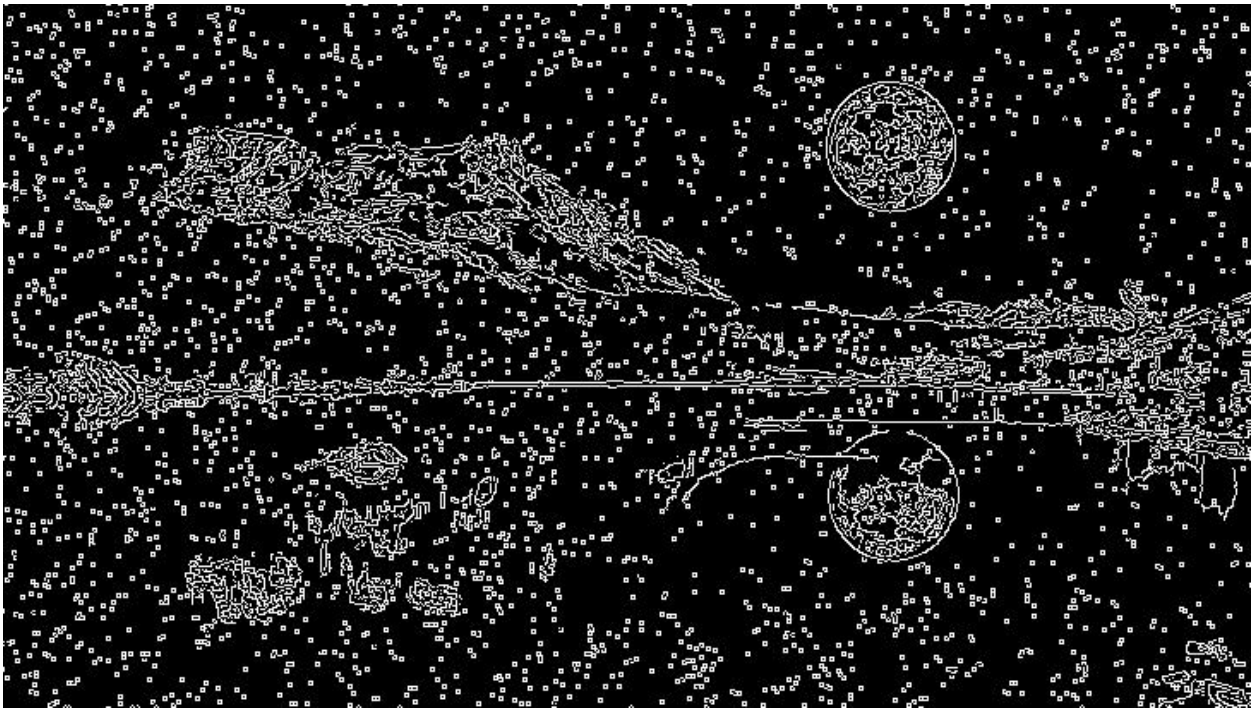


Figure 145: Output of Bilateral Filter after applying Canny at D = 29 of image 1

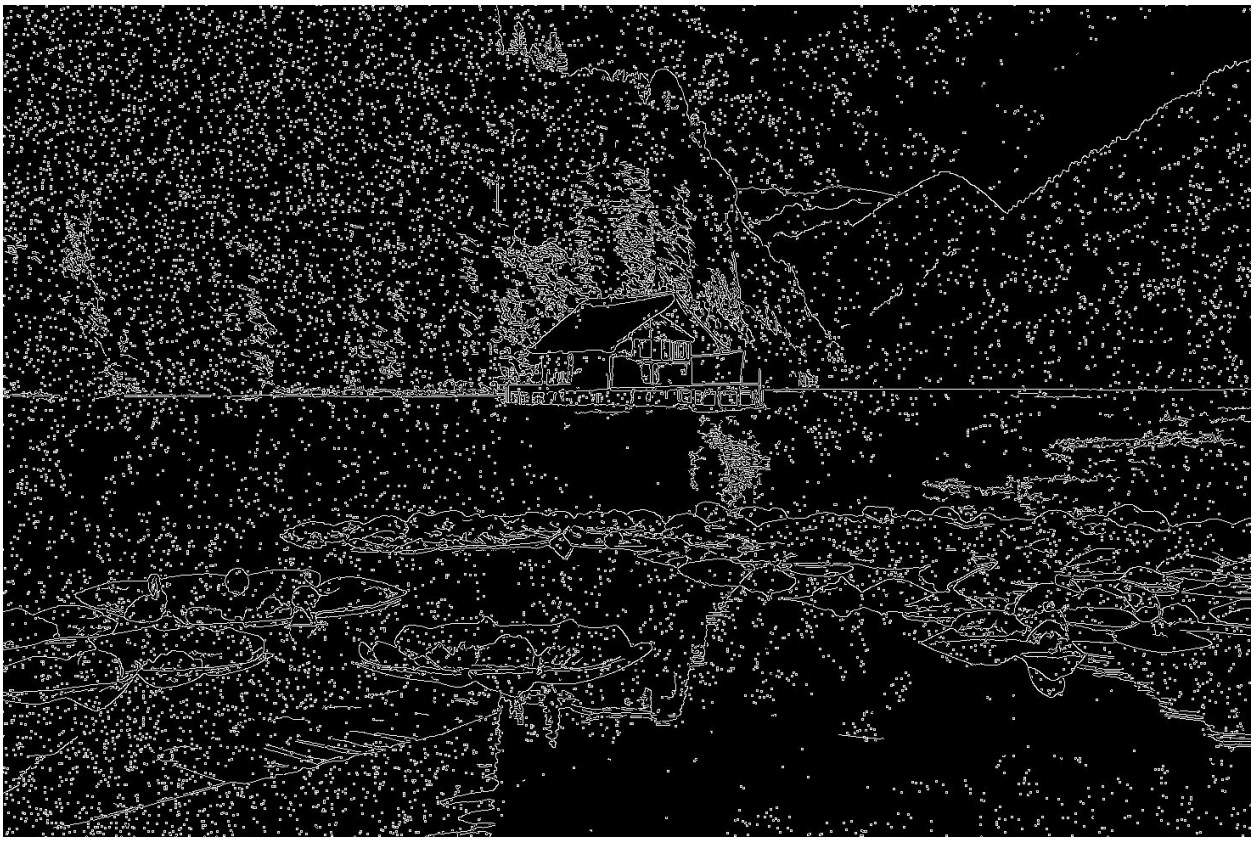


Figure 146: Output of Bilateral Filter after applying Canny at D = 29 of image 2

From the figures and table above the analysis as following:

1. Impact of Kernel Size on Noise Reduction (MSE and PSNR):

- **Box Filter**:
    - Increasing kernel size results in a noticeable decline in noise reduction performance. The MSE increases significantly (from 147.33 at K=3 to 310.02 at K=9), indicating more residual noise.
    - PSNR also decreases from 26.45 (K=3) to 23.22 (K=9), suggesting worse image quality as the kernel grows larger.
- **Gaussian Filter**:
    - Noise reduction is better than the Box filter at all kernel sizes, as evidenced by lower MSE and higher PSNR. For example, at K=3, Gaussian achieves an MSE of 110.81 compared to Box's 147.33.

- As kernel size increases, the performance degrades more gradually (MSE rises from 110.81 to 202.87, and PSNR decreases from 27.69 to 25.06).

- **Median Filter**:
  - This filter shows the best performance for noise reduction at smaller kernels, with the lowest MSE (104.99 for K=3) and relatively high PSNR (24.92).
  - For larger kernels (K=9), the MSE increases significantly to 291.63, and the PSNR drops to 23.48, showing a decline in quality.

- **Bilateral Filter**:
  - The Bilateral filter performs poorly for noise reduction compared to others for small diameters (D=9, MSE = 179.50).
  - However, as the diameter D increases, noise reduction improves (MSE decreases to 131.71 for D=29), showing its strength in smoothing while preserving edges.

2. Impact of Kernel Size on Edge Preservation:

- The Bilateral filter is the best at preserving edges due to its spatial and intensity-based weighting, evident in its increasing PSNR values from 25.59 (D=9) to 26.93 (D=29).
- The Median filter is also good for edge preservation at smaller kernels, but it struggles with larger kernels where blurring dominates.
- Gaussian and Box filters are less effective for edge preservation, as they both involve uniform smoothing across the neighborhood, which tends to blur edges.

3. Processing Speed (Computational Time):

- **Box and Gaussian Filters**:
  - Both are computationally very efficient, with processing times of 1–2 ms across all kernel sizes.

- **Median Filter**:
  - Also efficient, with similar computational times (1–2 ms) across kernel sizes.

- **Bilateral Filter**:
    - Computationally expensive, with processing time increasing as D increases (21 ms for D=9 to 210 ms for D=29).
    - This highlights a major trade-off between its superior edge preservation capabilities and processing speed.

Discussion

1. Noise Removal:

- **Simple Filters (Box, Gaussian, Median)**:
    - From the results, I observed that the **Box filter** performed the worst at removing noise, especially with larger kernel sizes. The MSE increased significantly as the kernel size grew, showing it struggles with both salt-and-pepper and Gaussian noise.
    - I found that the **Gaussian filter** provided better noise reduction compared to the Box filter. Its MSE and PSNR values were more consistent, particularly for Gaussian noise, but its performance slightly declined with larger kernels.
    - The **Median filter** showed excellent performance for salt-and-pepper noise. It had the lowest MSE for low and medium noise levels when small kernels were used, making it the most effective filter for this noise type.
- **Advanced Filter (Bilateral)**:
    - I discovered that the **Bilateral filter** was not as effective for high levels of salt-and-pepper noise. However, as the diameter increased, it performed much better for Gaussian noise, showing a strong balance between noise reduction and preserving quality.

2. Edge Preservation:

- **Box and Gaussian Filters**:
    - Based on my analysis, the Box and Gaussian filters caused significant blurring, especially with larger kernels, which led to poor edge preservation. Both filters were unable to maintain fine details in the image.
    - The Gaussian filter was slightly better than the Box filter at retaining edge sharpness, but neither was ideal for preserving details.
- **Median Filter**:

- I observed that the Median filter preserved edges well when smaller kernels were used, particularly for salt-and-pepper noise. However, as the kernel size increased, it began to smooth the image excessively, leading to a loss of fine details.

- **Bilateral Filter**:
  - It became clear to me that the Bilateral filter outperformed the others in preserving edges. It successfully smoothed the noise while maintaining sharpness and fine details, even with larger diameters. This was most noticeable with Gaussian noise.

3. Computational Efficiency:

- **Box, Gaussian, and Median Filters**:
  - From my experiments, I realized these filters are very fast, with processing times of only 1–2 ms, regardless of kernel size. This makes them highly suitable for real-time applications.

- **Bilateral Filter**:
  - I noticed that the Bilateral filter was much slower. The processing time increased drastically as the diameter grew, from 21 ms for a small diameter to 210 ms for the largest. This limits its practicality in time-sensitive scenarios.

4. Kernel Size Sensitivity:

- I observed that the filters reacted differently to changes in kernel size:
  - The **Box filter** was highly sensitive; larger kernels significantly reduced its ability to remove noise and preserve edges, making it unsuitable for large kernels.
  - The **Gaussian filter** was less affected by kernel size but still showed a drop in performance as the kernel got larger.
  - The **Median filter** was very sensitive, working best with small kernels for both noise removal and edge preservation. Larger kernels, however, led to oversmoothing.

- The **Bilateral filter** was the least affected by kernel size changes. Larger diameters improved noise reduction while still preserving edges, although this came at the cost of longer processing times.

5. Exploring Trade-Offs:

- From these experiments, I realized that there are clear trade-offs between noise reduction, edge preservation, and computational cost:
  - The **Box and Gaussian filters** are fast and work well for basic noise reduction, but they fail at preserving edges.
  - The **Median filter** is a strong choice for salt-and-pepper noise with small kernels, but it loses effectiveness at larger sizes.
  - The **Bilateral filter** strikes the best balance between noise reduction and edge preservation, particularly for Gaussian noise, but it is computationally expensive, especially with larger diameters.

Conclusion

In this assignment, I compared the performance of different filters, including Box, Gaussian, Median, and Bilateral, on noisy images with salt-and-pepper and Gaussian noise. Each filter had strengths and weaknesses depending on the noise type, kernel size, and processing time.

The Box filter was the fastest but had poor noise removal and edge preservation, especially with larger kernels. The Gaussian filter performed better for general noise reduction but caused some blurring. The Median filter was the best for removing salt-and-pepper noise and keeping edges sharp with small kernels, but it smoothed too much with larger ones. The Bilateral filter preserved edges the best, especially for Gaussian noise, but was much slower, making it less practical for real-time use.

Overall, the choice of filter depends on the task. For quick processing, the Gaussian filter works well. For salt-and-pepper noise, the Median filter with a small kernel is best. If edge preservation is key, the Bilateral filter is the best option if time is not a concern. This assignment helped me understand the trade-offs between noise removal, edge preservation, and speed when using different filters.